Efficient Object Recognition and Image Retrieval for Large-Scale Applications

by

John Jaesung Lee

S.B. Computer Science and Engineering, MIT 2007
S.B. Physics, MIT 2007

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Engineering in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 2008

Author .....
Department of Electrical Engineering and Computer Science
May 9, 2008

Certified by .....
Trevor J. Darrell
Associate Professor
Thesis Supervisor

Accepted by .....
Arthur C. Smith
Professor of Electrical Engineering
Chairman, Department Committee on Graduate Theses

Efficient Object Recognition and Image Retrieval for Large-Scale Applications

by

John Jaesung Lee

Submitted to the Department of Electrical Engineering and Computer Science
on May 9, 2008, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Electrical Engineering and Computer Science

Abstract

Algorithms for recognition and retrieval tasks generally call for both speed and accuracy. When scaling up to very large applications, however, we encounter additional significant requirements: adaptability and scalability. In many real-world systems, large numbers of images are constantly added to the database, requiring the algorithm to quickly tune itself to recent trends so it can serve queries more effectively. Moreover, the systems need to be able to meet the demands of simultaneous queries from many users.

In this thesis, I describe two new algorithms intended to meet these requirements and give an extensive experimental evaluation for both. The first algorithm constructs an adaptive vocabulary forest, which is an efficient image-database model that grows and shrinks as needed while adapting its structure to tune itself to recent trends. The second algorithm is a method for efficiently performing classification tasks by comparing query images to only a fixed number of training examples, regardless of the size of the image database. These two methods can be combined to create a fast, adaptable, and scalable vision system suitable for large-scale applications.

I also introduce LIBPMK, a fast implementation of common computer vision processing pipelines such as that of the pyramid match kernel. This implementation was used to build several successful interactive applications as well as batch experiments for research settings. This implementation, in addition to the two new algorithms introduced by this thesis, are a step toward meeting the speed, adaptability, and scalability requirements of practical large-scale vision systems.

Thesis Supervisor: Trevor J. Darrell
Title: Associate Professor

Acknowledgments

Many thanks are in order. I am glad to have had great mentors, colleagues, friends, and family members who have helped to guide and support me through the arduous labor of my undergraduate and graduate work at MIT.

I would like to thank Trevor Darrell for being a great advisor and mentor during my time at MIT. Having entered the group in my sophomore year as an inexperienced coder with only promises of motivation but little knowledge of computer vision, I stand here three years later with many accomplishments achieved under his supervision that I can be proud of. I thank him not only for the opportunity he gave me early on in my undergraduate career but also for the support and advice, technical or not, that over the years have armed me with the skills and knowledge to be successful in my future career.

I also owe many thanks to Kristen Grauman, with whom I worked closely with while I was an undergraduate at MIT. Under her guidance I gained a lot of knowledge about the field, developed great research habits, learned the values of a strong work ethic, and became inspired to aim high. She was always there to help whenever there was something I wanted to learn or didn't understand. Her dedication to success is extraordinary and it has shone through as she continues her successful career.

I have had the pleasure of working with Tom Yeh for the past couple of years, who has taught me many of the skills involved in being a good researcher. I enjoyed the trip to ICCV in Rio de Janeiro, where I got first-hand experience in a prestigious academic setting; Tom was, and continues to be, a great collaborator for many interesting projects.

My friends have been responsible for keeping me sane throughout these years despite the enormous workloads involved with MIT. They were always – literally 24 hours a day – there to take my mind off work that would have otherwise driven me crazy; they have been an amazing support group and a real pleasure to spend time with. Special mention goes to Justin, for teaching me the values of quality control; Ulzie, for making it look like I was always at Next House; Vince, for constantly re-

minding me of my middle initial; Zach, for somehow making a physicist out of me; Jelani, for demonstrating the importance of return values (and some grad school advice to boot); Ashish, Kenny, and Sharat for reebuh, reebuh, and reebuh (respectively); Sue-Ho, for bringing to my attention the significance of Building 7's mailing address; the upperclassmen, for keeping me motivated by calling me a noob whenever I was about to give up on something; the underclassmen, for the endless S-games; and to all of the members of my former living group, 3rd West, for they are the only ones who know the secret of page 77 of this thesis.

Finally, I would like to thank my parents and other family members for constantly supporting me through my life as a student at MIT. They have inspired me to keep going through the difficult times and taught me to always reach for the stars, never settling for anything less.

Bibliographic Notes

Portions of this thesis are based on the following papers:

Chapter 3 (joint work with Tom Yeh):
Tom Yeh, John J. Lee, and Trevor Darrell. Adaptive Vocabulary Forests for Dynamic Indexing and Category Learning. In *Proceedings of the IEEE International Conference on Computer Vision* (ICCV), Rio de Janeiro, Brazil, October 2007.

Chapter 4 (joint work with Tom Yeh):
Tom Yeh, John J. Lee, and Trevor Darrell. Scalable Classifiers for Internet Vision Tasks. To appear, *First IEEE Workshop on Internet Vision*, in conjunction with *Computer Vision and Pattern Recognition* (CVPR), Anchorage, Alaska, June 2008.

Chapter 5:
John J. Lee. LIBPMK: A Pyramid Match Toolkit. *MIT CSAIL Technical Report*, MIT-CSAIL-TR-2008-017, Cambridge, MA, April 2008.

Contents

List of Figures

Chapter 1

Introduction

The task of organizing all of the visual information available to us today is overwhelming. Digital imagery has become increasingly widespread thanks to the popularity of mobile camera-phones and online photo-sharing services such as Flickr[1] and Picasa[2]. Many images are made available via keyword-based search (Google Image Search, for instance, indexes billions of images), but image-based search (where queries are images rather than keywords) has a different set of computational needs. There is great demand for fast and space-efficient algorithms for recognition and retrieval, as well as system architectures which can scale to reliably handle the computational load of Web-scale image-based search.

The design of a system that can be deployed on a large scale, supporting many images and many users, presents many problems. In this chapter, I discuss the general challenges presented by object recognition and content-based image retrieval. I describe several interesting interactive applications of recognition and retrieval, and then highlight the new challenges faced when scaling these problems up. This is followed by an overview of my contributions to this line of research, which is a step toward solving these particular problems.

Object recognition is the problem of learning visual categories based on images, and then identifying the categories of new query images. In practice, users of object

[1] http://www.flickr.com
[2] http://picasa.google.com

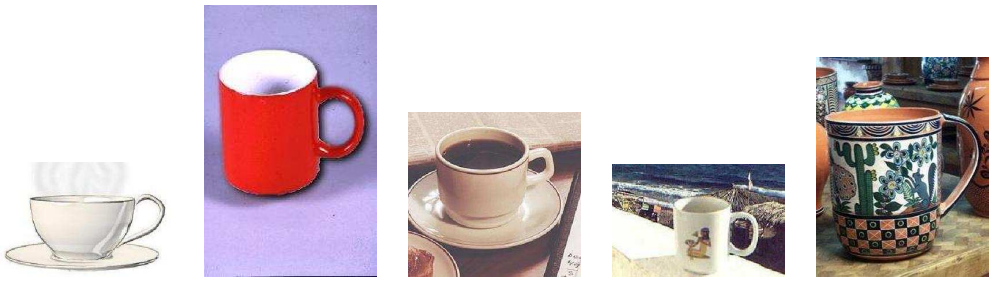


Figure 1-1: Example images of cups, taken from the Caltech-101 dataset [9]. While humans have no problem recognizing each of these images as cups, computers have trouble dealing with the wide variety of shapes and the background clutter.

recognition systems are interested in recognition at both the *category* and *instance* levels. At the category level, we aim to recognize objects of a conceptual class; for example, a home assistance robot that is told to "water the plants" would try to identify instances belonging to the generic "plant" category. On the other hand, the instance-level recognition problem is one in which the user wants to identify specific objects; this is akin to telling the home robot to water one particular plant.

In the content-based image retrieval problem, we seek to perform a search over a database of images, using an image as a query. Unlike traditional keyword-based searching, the output is sorted by some notion of its *visual* similarity to the query. There is a strong need for efficient indexing mechanisms: a database's usefulness is limited by its size, but also by its ability to perform its queries quickly for interactive applications.

In addition to speed, the ability of a recognition or retrieval system to perform *accurate* visual similarity searching is important. Being able to judge visual similarity with accuracy comparable to that of humans is a daunting task; it is clear that humans generally have no problems recognizing objects in various poses, under different lighting conditions, and in the presence of clutter or occlusion. As images are represented by computers as a 2-dimensional array of pixels, even a slight change in lighting or pose can completely change the array of pixels. For object recognition specifically, objects may also exhibit wide intra-class variation (see Figure 1-1).

1.1 Interactive Vision Applications

In this section, I describe three interesting possible applications of an Internet-scale vision system: example-based photo grouping, content-based tag suggestion, and category-based image search. These systems all highlight the need for particular design requirements:

- **Speed**: users want results quickly and sometimes do not mind reading beyond just the top result
- **Adaptability**: user preferences, styles, and habits change over time
- **Scalability**: many users all access the system at the same time

Below, I give example usage scenarios of each application and discuss the requirements outlined above in each context.

Example-based photo grouping

Example-based photo grouping enables Internet users to organize their online photos into visually coherent groups. Users can create "folders" and select several example photos that indicate what kinds of photos should go into each folder. Figure 1-2 shows a screenshot of a prototype system offering this feature. In this particular scenario, a user has created two folders, labeled *gun* and *zebra*. By dragging-and-dropping new photos into a designated area on each folder, the recognition system suggests other images that may also belong to that folder. As this is a real-time application, it requires an extremely fast response from the system because the updates to the user interface need to occur as the user is moving the photos around. Additionally, it highlights the need for adaptiveness: as users add new example photos to a folder, the system's output should change quickly to reflect the new data.

Content-based tag suggestion

Content-based tag suggestion helps Internet users tag their online photos quickly. Figure 1-3 illustrates a prototype system that shows a typical scenario of content-

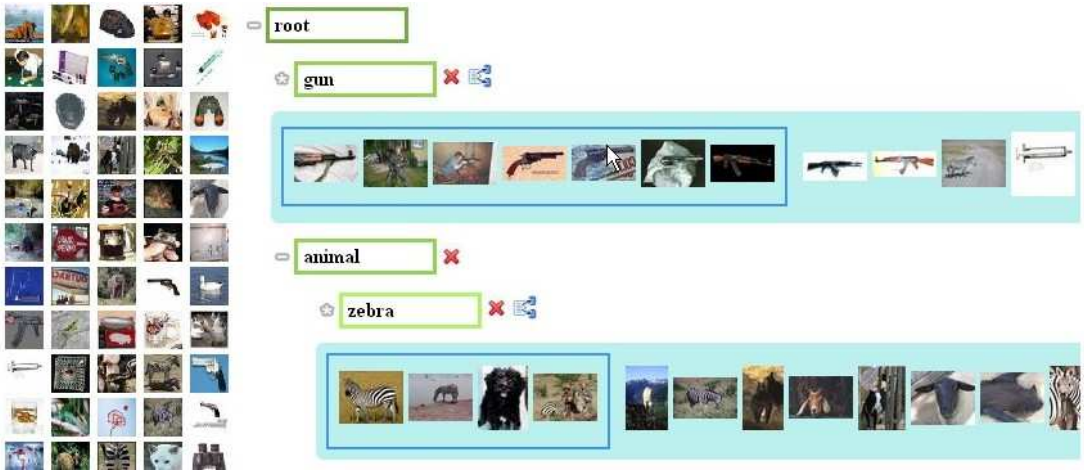


Figure 1-2: Example-based photo grouping

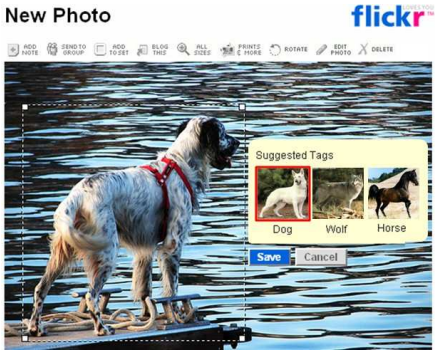


Figure 1-3: Content-based tag suggestion

based tag suggestion. First, a user uploads a photograph to an online photo album; the recognition system then analyzes the image content and suggests some category labels (in the example shown, the system has suggested *dog*, *wolf*, and *horse*) it considers to be visually relevant to the uploaded photograph.

The suggestions made by the system can be based can be based either on a global classifier built over all images and all users, or on individual classifiers that are tuned for that particular user based on other images that the same user has uploaded. The system's adaptability is important in this situation: certain kinds of images may suddenly become popular over a short period of time, so the recognition should be tuned to match those trends. Scalability is also of great importance, since aggregate classifiers built with data from thousands of other users are used in the recognition task.

Figure 1-4: Category-based image search

Category-based image search

Category-based image search allows Internet users to search for Web images using a set of images instead of keywords a the query. The images in the query set collectively define a new category of images the user intends to find. Figure 1-4 shows a screenshot of a prototype category-based image search engine that manifests itself as an extension of an existing keyword-based image search engine. In this particular example, a user has uploaded multiple query images as examples of the category sought (*blimp*). With the assistance of the recognition system, the search engine returns a result page containing Web images that are likely to belong to the same category.

This application is similar to existing content-based image retrieval systems in that it fetches images using other images as the query. The key difference is that the search results in this application are not retrieved only by their visual similarity to the user's query image, but also by the category label an object recognizer assigns to them, thus allowing the search engine to leverage category information from other searches and databases. Clearly, such a system must be able to scale up to the demands of many users at once, and needs to be able to provide its search results as quickly as keyword-based image searches.

1.1.1 Problems with current vision algorithms

Many computer vision systems today are generally evaluated assuming that the results are to be used by robots or automatic multimedia archivers. However, applications

such as those described above are designed for human users, whose evaluation of the quality of the system may differ in several ways. First, the user experience is affected greatly by delicate speed-accuracy trade-offs, whereas machines will not complain about a computation that takes several hours longer but provides a marginal boost in accuracy. Second, as humans are generally better at classifying images than computers, they can quickly detect when their object recognizer has made mistakes, and are thus available to correct the mistakes.

Because most current recognition and retrieval algorithms are not necessarily designed for use specifically by Internet users, they are generally evaluated by their accuracy and not their ability to perform dynamic operations in interactive environments, thus imposing a different set of requirements. This thesis addresses two of those major requirements: adaptability and scalability.

Adaptability and dynamic environments

Many real world problems are dynamic in nature: the set of images and captions in news stories changes with current events; the set of objects to be recognized by a home robot depends on the particular home it inhabits; personal media to be organized visually will depend on the recent experiences of the user. It is rarely the case that a single, fixed vocabulary is optimal in these environments. Thus, a user-oriented vision system should be able to tune itself to the changing interests of its users. Because the focus of user attention can shift rapidly, it is essential that the adaptation computations be performed online. It is costly to rebuild a vocabulary and recompute the bag-of-words representations for every image whenever a new image is added to the database. Although the amount of offline computation does not affect the user's query time, the system as a whole will suffer if the database is frequently out-of-date due to long offline computations.

Scalability and speed

Large-scale systems are queried very frequently with new images. Because the the training steps (e.g., building vocabularies, populating an index, or constructing clas-

sifiers) can be performed offline, the total time spent processing new query images from users can easily exceed the total time spent in the offline step. As a result, we should place a greater emphasis on the query time when evaluating user-targeted vision systems. The training needs to be performed only once per database update, whereas the total time spent on query images scales with the number of users. If the query time for a particular algorithm is low enough, it can generally be made to scale easily with the number of users, simply by adding more machines to the computation. In this setting, a copy of the trained model is kept on each of many servers, and single queries are distributed between them. This, however, becomes problematic if the trained model is too large to fit on a single machine. In addition to scaling with the number of users, it is essential that practical systems be able to also scale with the number of images and classes in the database.

Many current algorithms for recognition and retrieval have clear speed-accuracy trade-offs; reducing the number of extracted local features (e.g., by sampling them) speeds up all steps of the process, but will also decrease recall and reduce the discriminative ability of classifiers. A human's evaluation of a vision system can be very sensitive to time, especially for interactive applications, At the same time, humans can also tolerate small errors: a correct entry in the "first page" of results is nearly as good as one appearing as the top result. Users may also want to see a group of results rather than one of them, just to see what else the system might suggest. This tolerance for small errors that we may be able to bias the speed-accuracy trade-off to produce faster output.

Requirements of interactive vision applications

As discussed above, some major problems posed by large-scale interactive object recognition and image retrieval systems are that they must (1) respond quickly to user queries, (2) scale well with growing numbers of users and images, and (3) adapt to recent trends in the data as new images are added. This thesis provides a step towards solving these three problems. In the following section, I describe my work in each of these areas and give an overview of this thesis.

1.2 Thesis Overview

1.2.1 Contributions

In this thesis, I develop a new efficient algorithm that incrementally computes sets of vocabulary trees. This algorithm maintains both an inverted index and histogram pyramid representations, allowing for more versatility in that it can be used both for recognition and retrieval tasks. Unlike previous methods, this algorithm also allows for fast online adaptation so that new images can be quickly added to the database for immediate retrieval. On dynamic recognition tasks, where the categories and images under consideration may change over time, this adaptive vocabulary-tree method offers not only speed improvements, but significant gains in recognition accuracy as well.

For the task of object recognition, I also describe a new method of approximating SVMs to reduce the query time experienced by users. A standard SVM computes the similarity between the query image and every support vector, which can be expensive. By combining content-based image retrieval techniques with object recognition methods, we can quickly retrieve a ranked list of relevant support vectors and choose to compute only as many similarities as we need in order to produce output in a timely fashion. This method produces nearly the same accuracy as a standard SVM, but with a fraction of the computation time.

My research in this area has led me to develop LIBPMK [15], a fast open-source[3] implementation of the Pyramid Match Kernel [10], on which the implementations of the two algorithms described above are built. LIBPMK provides functionality for every stage of the recognition pipeline, from the raw images all the way up to SVM training and evaluation, and can operate orders of magnitude faster than current alternatives. This code was used to build several successful interactive applications. One of the applications in particular allows users to upload photos to the system via a camera phone. The system adds the new image to the database and performs a query to retrieve a ranked list of similar images in real-time.

[3]Code, documentation, and examples are available at http://people.csail.mit.edu/jjl/libpmk

1.2.2 Outline

In the following chapter I give a detailed description of current object recognition and image retrieval systems, and describe work related to their scalability and adaptability. Then in Chapter 3 I describe an online recognition/retrieval system that tunes itself to trends as new images are added to the database, and describe the speed-accuracy trade-offs involved. In Chapter 4 I describe a scalable architecture suited for Internet-scale object recognition with fast approximate SVMs using vocabulary trees. Chapter 5 is devoted to the details about the implementation of LIBPMK and applications that were built upon it.

Chapter 2

Background and Related Work

2.1 Vocabulary Models

Recent work has shown that breaking up images into component parts can help algorithms be robust to the aforementioned variance encountered in real-world images. Images are represented as sets of *local features*, each of which is a vector describing a small patch of the image. The images can then be compared on the basis of the similarity between their local feature sets. Figure 2-1 shows how two images can be compared by extracting local features and matching the features from one image to those from the other image.

The "bag-of-words" approach, inspired by text retrieval methods, was introduced in [25]. The bag-of-words methods typically apply clustering techniques on large groups of local features from many images, forming a *vocabulary* of "visual words" by partitioning the feature space. Thus, each image is represented simply as a set of these discrete words, giving these methods the advantage of reducing images to a compact histogram representation that can then be applied to a number of learning methods depending on the task. In object recognition, for example, we may use a support vector machine (SVM) to build classifiers in the histogram space; for retrieval, we may build an index that maps visual words to images, similar to the way traditional keyword search engines index text documents.

Hierarchical vocabulary models were later introduced to address the issue of scal-

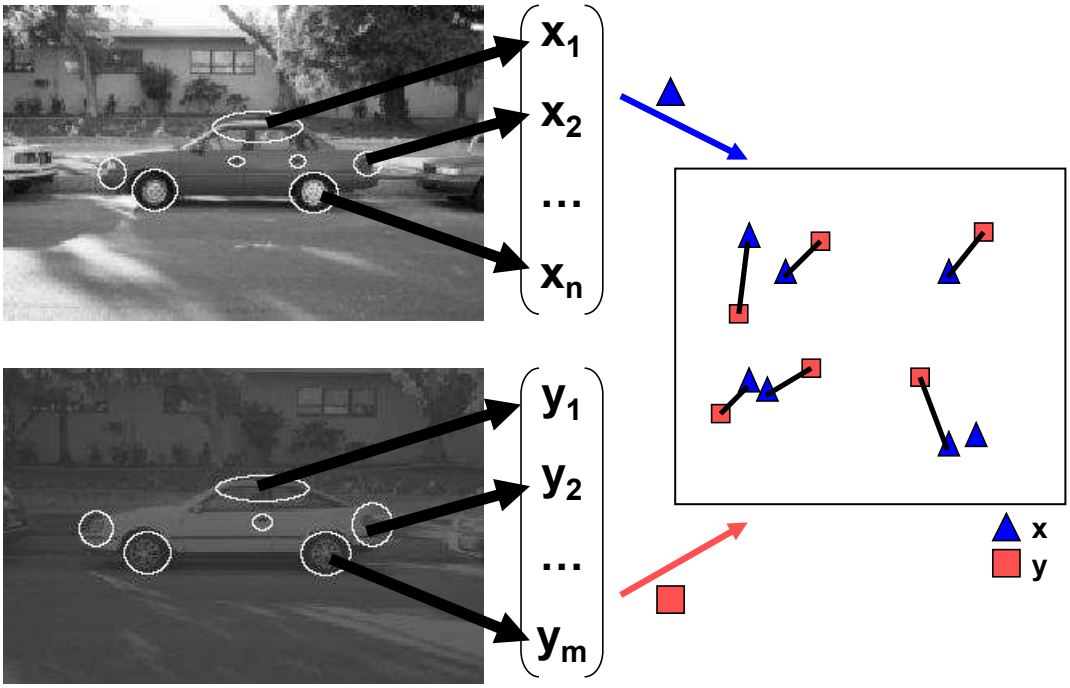


Figure 2-1: A toy example showing local features extracted from two images. A local feature descriptor is extracted for each circled region, and each image is represented as a set of these descriptors. On the right, a similarity computation is performed by pairing the points in the local feature space from each image.

ability. As more words are introduced learned, the computational complexity of recognition and retrieval systems increases dramatically. For recognition, histogram pyramids [10] have proven valuable for fast similarity computations while compromising very little in recognition accuracy; for retrieval, vocabulary trees [21] are used to arrange the visual word index hierarchically.

2.2 Hierarchical Models

Recognition methods based on local feature models [17, 19] have been demonstrated to have success across a range of tasks. However, search complexity with local feature models can be costly. To speed up comparisons over large databases, histograms

over feature vocabularies were introduced in [25, 11] and further demonstrated that a feature pyramid could approximate a correspondence-based distance measure. Efficient computation of vocabulary models was addressed by [21], who showed that hierarchical k-means could quickly compute very large vocabularies. There is a close relationship between the vocabulary tree of [21] and the feature pyramid of [11], although they are used for different computations. This section describes those two approaches and their applications.

2.2.1 Vocabulary tree

A vocabulary tree is an efficient data structure for indexing images based on visual words. When the number of visual words is large, instead of having to scan through the entire vocabulary to find matching images, a tree structure allows vocabulary lookups in sub-linear time.

Given a large number of features extracted from training images as the input, the vocabulary tree is learned using hierarchical k-means. The learning is controlled by two parameters: the number of levels L specifies the height of the resulting tree, and the branch factor B specifies the number of children each node has. A leaf node of the tree is called a *visual word*: as in the traditional bag-of-words models, it represents a small local neighborhood of the feature space. The hierarchical layout of this structure allows us to quantize features very quickly compared to the flat bag-of-words representations. The layout of the tree is illustrated in Figure 2-2.

Each visual word in the tree is associated with an inverted file that stores pointers to training images that contain that visual word. It also keeps track of the number of times each visual word appears in each image. Given a query image q represented as a set of feature points, images similar to q can be quickly found by looking up the images associated with visual words corresponding to q's features, and then ranking these images by having each word cast a vote. Voting methods such as this are suitable for fast content-based image retrieval.

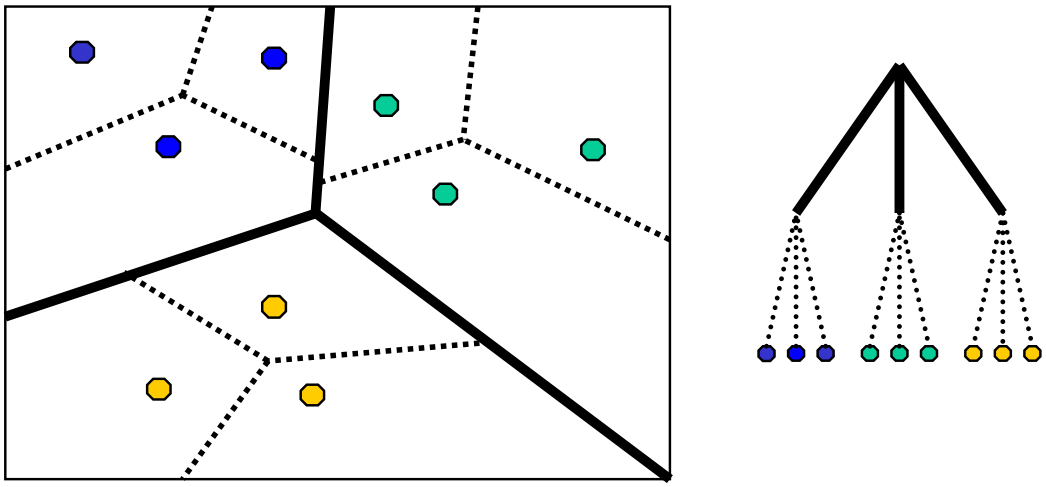


Figure 2-2: The left image shows how a vocabulary tree with $B = 3$ and $L = 2$ might partition the 2-dimensional feature space consisting of 9 points. The corresponding tree representation (hierarchical bag-of-words model) is shown on the right.

2.2.2 Pyramid match

The vocabulary-guided pyramid match kernel (PMK) method, introduced by [11], is a fast method of approximating the minimum-cost partial match between two sets of high-dimensional vectors. It makes use of a hierarchical vocabulary to define the placement and sizes of the histogram bins. The similarity score between two pyramids is computed by matching points in a bottom-up fashion, weighting new matches at each level of the pyramid inversely proportionally to the bin size.

To generate a pyramid from a vocabulary tree and a new image, the points from the image are first embedded into the tree; each point traces a path down the vocabulary tree. For each such path, we create a parallel path in the pyramid. This way, the pyramid maintains a sparse representation, and similarity computations between two pyramids can be performed with one pass through each pyramid.

The approximate matching score between two pyramids is computed with a weighted intersection measure. At a fixed level in the vocabulary tree, two points are said to match if they fall into the same bin. Because the number of matches in a particular bin includes all of the matches found at descendants' bins, we find the number of *new* matches at a bin by subtracting the number of matches found in the bin's chil-

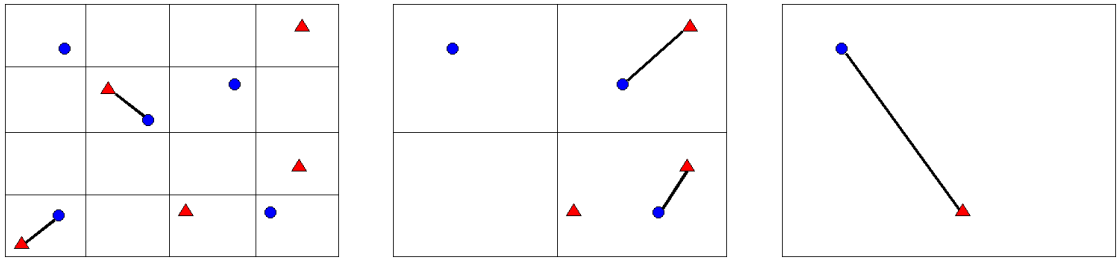


Figure 2-3: The pyramid match efficiently computes approximate partial correspondences by dividing the feature space repeatedly and propagating unmatched points to the next highest level. Shown here is a 3-level pyramid match. The features shown are partitioned by uniform grids, but the vocabulary tree approach is applicable here as well.

dren. Beginning at the lowest level of the pyramids, we add to the matching score a weighted count of the number of new matches present in each bin. The weight associated with each bin is inversely proportional to the size of that bin. An example of how a pyramid match score is computed is illustrated in Figure 2-3.

The vocabulary-guided pyramid match is useful because it allows us to match sets of high-dimensional vectors in linear time. When the weights are chosen to be inversely proportional to the size of the bin, we are guaranteed to have a kernel that satisfies Mercer's condition [11]. The pyramid match method has been shown to be both fast and accurate for many object recognition tasks.

2.3 Current recognition algorithms

In this section, I give an overview of some interesting current recognition algorithms relevant to the work presented in this thesis. The methods described here—the spatial pyramid match, the PHOG, and SVM-KNN—were shown to achieve state-of-the-art recognition performance. Outlined below are brief descriptions of each of these methods and a discussion of how they might behave differently in a large-scale user-oriented environment.

2.3.1 Spatial pyramid match

The spatial pyramid match method, introduced in [14], is a natural extension to the pyramid match. The standard pyramid match computes approximate correspondences in the feature space of a pair of images, which may or may not include spatial information. The spatial pyramid match, on the other hand, computes match scores based explicitly on the position of features relative to the image boundaries; to do this, it builds multi-resolution histograms over the 2-dimensional *image* space rather than the feature space, thus penalizing matches between corresponding features in a pair of images whose positions are not near each other.

The features of each image are first quantized into hundreds of visual words, ignoring their position in the image. Then, for each unique word, a pyramid match is computed using the x-y coordinates as the features, and the score is accumulated across all channels. This method was shown to achieve good performance on the Caltech-101 [9] database. This database, however, has often been criticized because the objects in most images are usually in the center of the image. The spatial pyramid match takes advantage of this property to boost recognition accuracy.

The main speed bottleneck in this method is the same as with any bag-of-words method: clustering the features in the training set to build a vocabulary requires heavy computation. However, this load is present only during the training step; when querying, the features need only be quantized. Additionally, the feature space used by the pyramid match computation is bounded by the image size. Because image sizes are easily normalized, this means that only a small number of pyramid levels due to the small size of the feature space.

2.3.2 Pyramids of histograms of oriented gradients

Further work in hierarchical schemes for similarity computations was introduced by [3] with a descriptor for representing the shape of an object in the image. Rather than working with local features, they use a simple edge detector and use the edge gradients as the underlying representation. For any given subwindow of the image, a

histogram of oriented gradients (HOG) accumulates, over each edge point, the angles of the gradients that fall into a particular range. The contribution of an edge point to the HOG is weighted by the gradient magnitude at that point.

For efficient matching, it is natural to apply the pyramid method to this by imposing grids of various resolutions on the image space. The resulting descriptors, the pyramid of HOGs (PHOG), can be compared using the same histogram intersection method as the pyramid match, or by other means such as the χ^2 distance between the histograms at each level. This method was also shown to achieve good recognition accuracy on the Caltech-101 database. Like the spatial pyramid match method, this takes advantage of the fact that the objects in the image are generally well-centered in the image; there may be losses in performance when the objects are off-center.

The PHOG algorithm is fast because the features are scalar edge orientations, which are quantized easily without the need for clustering or expensive preprocessing. In addition, the pyramids are simple to generate because they are made in the image space, as with the spatial pyramid match.

2.3.3 SVM-KNN

SVM-KNN was introduced in [30] as a way of quickly building small classifiers in local neighborhoods in the image similarity space for object recognition. Rather than training any SVM classifiers in advance, they instead opt to perform classification in two stages. In the first stage, they apply a "crude" (presumably faster) distance function that can quickly fetch the k images in the training set that are most similar to the query image. Once the nearest neighbors are retrieved, a new multi-class SVM is trained on-the-fly using only those k images as training data using a more "accurate" distance function, which is then used to classify the query image.

The SVM-KNN method attempts to avoid the scaling problem by not requiring any processing during the training phase of the algorithm. Furthermore, the only computation that applies to the entire training set is the "crude" distance function which is used only to approximate the nearest neighbors; any further computations are done only in the local neighborhood, which is of fixed size k. This method was

shown to outperform both a simple nearest-neighbor classifier and a standard multi-class SVM.

This method faces problems, however, when scaling up to Internet applications. Despite being able to use a crude-but-fast distance function, the fact that it must recompute these distances for every query over every training image remains, making it slower as more users insert more images. Additionally, a new SVM must be solved for each user query. Although this method does not require an offline training stage, most of the work is done at query time. As discussed in Chapter 1, we would prefer to keep query time to a minimum for user-oriented applications.

2.4 Incremental Models

In the information retrieval community, the need for incremental clustering methods for dynamic environments has long been identified [7, 6, 31, 22]. Given the extremely high rate of update required by today's online information systems, it is important to pursue an incremental approach to updating clusters without having to perform complete re-clustering. Several deterministic and randomized incremental clustering algorithms were proposed in [6] with the goal of minimizing the number of clusters while enforcing a bound on the cluster diameter. However, their methods are not designed specifically for hierarchical clustering. In [22], an incremental algorithm was developed to maintain bottom-up binary decision trees for hand-written digit recognition. A more sophisticated bottom-up hierarchical clustering method that allows for varying branch factors was introduced in [31]. This method first incrementally builds a "clustering feature tree" with a single scan of the data, and then applies global clustering to produce a small number of clusters. Finally, in [7], an incremental clustering method was created for the task of classifying Web documents into a "document cluster tree" to support efficient lookups. This method updates the tree dynamically by splitting over-crowded leaf nodes. Despite serving the same goal—indexing—a document cluster tree is similar to a vocabulary tree only in that both represent documents (or images) as a bag-of-words. However, they differ in that the

former is binary and stores each document as a whole at a leaf node, whereas the latter is multi-branched and keeps each image as words distributed across several leaves.

These incremental methods, though effective for their respective target tasks, are not readily applicable for a vocabulary tree for three reasons: (1) a vocabulary tree is constructed top-down by hierarchical k-means, (2) a vocabulary tree is not a binary decision tree, and (3) a vocabulary tree represents images at leaf nodes per-feature based on an inverted filesystem. The algorithms described in Chapters 3 and 4 develop incremental methods designed specifically for vocabulary trees to give significant speed boosts to object recognition and content-based image retrieval.

Chapter 3

Adaptive Vocabulary Forests for Dynamic Environments

3.1 Overview

Histogram pyramid representations computed from a vocabulary tree of visual words have proven valuable for a range of image indexing and recognition tasks. However, they have only used a single, fixed partition of feature space. This chapter presents a new efficient algorithm to incrementally compute set-of-trees (forest) vocabulary representations, and shows that they improve recognition and indexing performance in methods which use histogram pyramids. This algorithm incrementally adapts a vocabulary forest with an inverted filesystem at the leaf nodes and automatically keeps existing histogram pyramid database entries up-to-date in a forward filesystem. It is possible not only to apply vocabulary tree indexing algorithms directly, but also to compute pyramid match kernel values efficiently. On dynamic recognition tasks where categories or objects under consideration may change over time, adaptive vocabularies offer significant performance advantages in comparison to a single, fixed vocabulary.

3.2 Motivation

It is rare that static representations are optimal in a dynamic world, yet most existing image indexing and category recognition models use a single fixed vocabulary representation. These models models typically apply a batch clustering technique to a subsample of training data and perform vector quantization before storing any image records or computing category models. Hierarchical vocabulary representations such as the vocabulary tree [21] and histogram pyramid [11] have recently proven valuable for a range of image indexing and recognition tasks, but also use a single, fixed vocabulary model.

In dynamic, real-world problems, user attention may shift to new classes of objects over time. While it is well accepted that model parameters should evolve as categories change dynamically, it is generally presumed that a single, fixed visual word vocabulary representation–if large enough–will suffice for all tasks. On one hand, a very large (but fixed) vocabulary avoids the need to ever recompute vocabularies when new images are added to the database. There are two problems with this approach, however. First, at any given time, only a small fraction of the enormous vocabulary is ever used. Second, the performance of a fixed vocabulary suffers if it is ill-tuned to the test distribution. Two categories may inhabit the same region of the feature space but prefer different ways of partitioning it. Rather than compromising, an adaptive tree offers the opportunity to choose the optimal partitioning for one of those categories if the other category is irrelevant to the user.

This chapter proposes an incremental vocabulary tree model that grows and adapts as new images are added to the database, based on a measure that encourages splitting nodes in the tree that become too ambiguous and pruning nodes that capture distinctions no longer relevant to the current set of tasks the system is facing (Figure 3-1). The computational cost per frame of our adaptive algorithm is minimal, and the resulting trees that are computed are similar in structure and performance to those computed offline.

Because hierarchical clustering algorithms can lead to irregular partitions of the

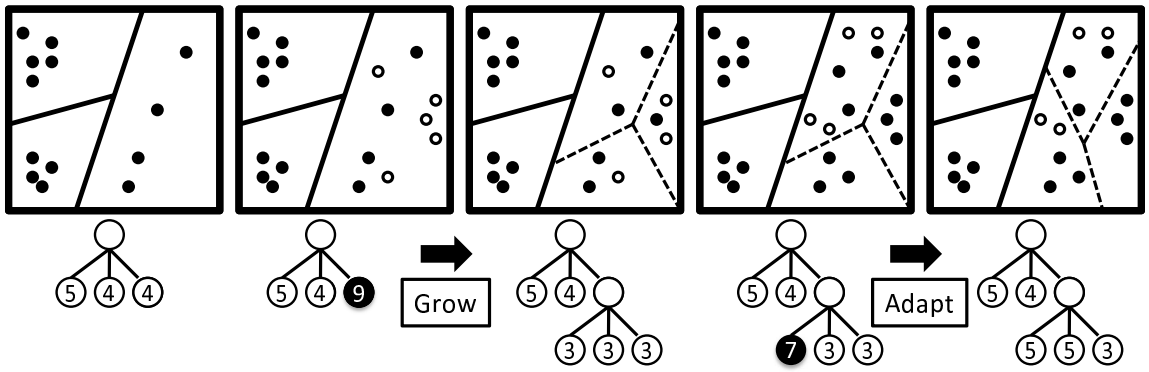


Figure 3-1: An adaptive vocabulary tree grows and adjusts partitions continuously as new feature points (shown as hollow circles) are added.

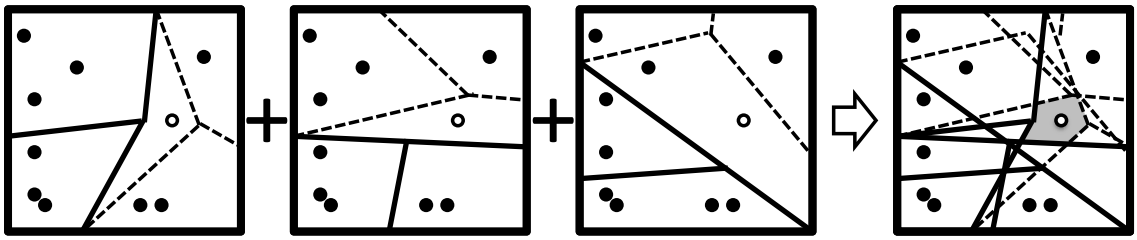


Figure 3-2: An ensemble of vocabulary trees can provide finer partitions of the feature space than a single tree.

feature space, we adopt a set-of-trees (forest) approach with an ensemble of quantization grids (Figure 3-2). Randomized forest classification models have been recently shown to have theoretical and practical advantages in learning and vision tasks, and to improve performance with voting-based [16] or SVM-based methods [20]. The incremental method for computing a set-of-trees vocabulary representation, developed in this chapter, is applicable to those based on L1-norm over histograms, as well as those which compute a value based on histogram intersection across multiple resolutions (e.g., the pyramid match kernel).

Moreover, this method includes a new algorithm for efficiently maintaining pyramid representations using an inverted filesystem storage model. Using this method, a set of pyramid match representations (or kernel values) can be computed by simply traversing the maintained pyramids. In this way, we can maintain an inverted filesystem (vocabulary tree) for fast retrieval, and can quickly update a forward filesystem

(pyramid match) model to compute image similarities.

In the following section I describe the foundations of vocabulary tree and pyramid match representations relevant to this method. I then develop a vocabulary forest model for these hierarchical vocabularies, and an incremental update algorithm which can keep current both an inverted (vocabulary tree) and forward filesystem (pyramid match) representation. In Section 3.4 I present experimental results and conclude with a discussion of this method's merits.

3.3 Algorithm

3.3.1 Growing new vocabulary trees

The tree growth method works in three steps:

1. A new point is inserted into the vocabulary tree;

2. Overcrowded leaf nodes (if any) get removed in preparation for restructuring;

3. The points displaced by the previous step are-clustered and inserted back into the vocabulary tree.

The behavior at each step is controlled by three parameters: the *capacity constraint* C which defines the maximum size of a leaf node, the *neighborhood size* S which sets the maximum number of points involved in any re-clustering step, and the *reduction factor* R which controls the maximum number of points allowed in a leaf node after a re-clustering step.

Each parameter serves a different purpose. First, for each node in the vocabulary tree, having more data points means that it has higher entropy. The capacity constraint C ensures none of the nodes can have more than a certain number of data points. Second, when forced to update the clusters, many existing algorithms, like [7], split only one cluster at a time, which can be too constrained. The neighborhood size S allows one to define an arbitrarily large neighborhood to update each time.

Finally, to prevent excessively frequent updates to the same part of the tree, we introduce the reduction factor R to make sure that newly created node will always have a reasonable amount of unused capacity to accommodate future data points. In Section 3.4.4, I show that although these parameters provide varying time-space tradeoffs, the recognition performance remains fairly insensitive to them.

In step 1, we insert a new point from an image into the tree by placing it into the node whose center is closer to the point than any of the node's siblings. We update the sizes of the nodes (if necessary) by recording the largest distance to the cluster center from any of the points in the node. In addition, we update the relevant pyramids to reflect the insertion of the new point (see Algorithm 1). If, as a result of the insertion, the number of points at the leaf w exceeds C, we trigger step 2 of our algorithm, which begins a process of *releasing* points in the neighborhood of the overcrowded node into a temporary pool Φ.

In step 2, we release a number of nearby points no greater than the neighborhood size S (Algorithm 2). We take a greedy approach that starts by removing the visual word w that triggered the update, and continues by removing w's siblings one by one (from the closest to the furthest from w). Whenever a visual word is removed by this process, its points are also released into Φ. If there are no siblings left, and the pool of released points has not yet reached size S, the algorithm moves to the previous generation and begins removing w's parent and its siblings. During this removal process, however, we ignore *unreleaseable* nodes whose descendants contain too many points to avoid pushing the size of Φ past size S. This remove-and-release process continues until all neighboring removable nodes have been removed.

Remark 3.3.1. *If all nodes containing more than $\frac{S}{B}$ points are marked as unreleaseable, then $|\Phi|$ is always at most S.*

Proof. Let Q be the minimum number of total points held by a node's descendants for the node to be considered unremovable. Then all removable candidate nodes must have at most $Q-1$ points underneath them. Consider the node x at the highest level reached by the removal algorithm (i.e., the lowest common ancestor of all removed

Algorithm 1 Insertion of a new point p from image i

```
procedure INSERT(p, i)
    w ← p's matching visual word
    Add p to the inverted file at w
    Δ_i ← the ith pyramid
    INSERTINTOPYRAMID(p, Δ_i)
    if |w| > C then
        Φ, x ← RELEASEPOINTS(w)
        RE-CLUSTER(Φ, x)
        P_Φ ← set of all pyramids with points in Φ
        UPDATEPYRAMIDS(Φ, P_Φ, x)
    end if
end procedure
```

nodes). Because every node can have at most B children, x has at most B siblings which were also removed. Thus, the maximum number of released points must be $B(Q-1)$. If we set $Q = \frac{S}{B} + 1$, we can ensure that we do not release more than than S points. □

In step 3, we begin the re-clustering process on the released points Φ. Let x be the lowest common ancestor of all removed nodes (i.e., the node at which the removal process stopped). Suppose x has m children remaining after the removal process. We run k-means clustering on Φ with $k = B - m$ to keep the branch factor constant throughout the tree. The reduction factor R is a ratio which specifies the constraint on the resulting distribution: if, as a result of the clustering, any of the nodes have more than $R \times C$ children, we recursively perform more levels of k-means clustering rooted at that child until the constraint is satisfied.

Normally we retrain a new sub-tree using the same height as the original one and hope that retraining can redistribute the points more evenly. However, there are times when the new sub-tree may not be able to provide the desired reduction factor when it is fixed to the same height. In such situation, we allow the sub-tree to increase its height by one in order to acquire the needed capacity to satisfy the reduction factor. Note that the height increase occurs only locally. Therefore, the resulting vocabulary tree may have some branches longer than the others.

Algorithm 2 Release from a visual word w's neighborhood at most S points for re-clustering

```
function ReleasePoints(w)
    x ← w                                   ▷ maintain x to be root of all released nodes
    Φ ← x's points
    while |Φ| ≤ S and x is removable do
        Y ← x ∪ {x's siblings sorted by distance to x}
        for all y ∈ Y such that |y| < S/B + 1 do
            Φ ← Φ ∪ y's points
            remove(y)
        end for
        if x is removable and x has a parent then
            x ← x's parent
        end if
    end while
    return (Φ, x)
end function
```

Algorithm 3 Cluster points in Φ and graft new nodes to x

```
procedure Re-cluster(Φ, x)
    m ← number of children of x has
    clusters ← do k-means on Φ with k = B − m
    for all c ∈ clusters do
        Add a new child y to x
        Add c's points to y
        if |y| > R ∗ C then
            Re-cluster(c's points, y)
        end if
    end for
end procedure
```

3.3.2 Pruning obsolete nodes

As new feature points are constantly inserted into a vocabulary tree, at any given time period, some of the tree nodes may take on many more points while others remain unused. A prolonged period of inactivity at a node indicates that the node is obsolete with regard to the current focus of the problem in a dynamic environment. To remove obsolete nodes, we maintain a list of access records to remember the least-recently used node: each time a node is accessed (i.e., a point is added to the a node), the node is moved to the front of the list. We specify a limit on the number of leaf nodes

a tree can have. If the tree ever grows beyond the limit, we prune the least recently used leaf nodes until the limit is met. The data presented in Section 3.4.5 show that a pruned tree can improve performance on dynamic recognition problems.

3.3.3 Pyramid maintenance

In this section, I describe how our method keeps the pyramid representations of images up-to-date, stored in a forward filesystem. Note that maintenance of this representation is not necessary for the vocabulary tree to function normally, but is rather an optional component which, when included, can immediately return any of the maintained pyramids corresponding to the current vocabulary. Since every bin in each pyramid corresponds to a node in the vocabulary tree, we maintain at each node a list of pointers to the bins in the pyramids they correspond to. This list is sparse; not every bin will contain a point from every image. When we retrieve the pyramid for a particular image, the bins report their sizes by following the pointer back to the vocabulary tree, which keeps the sizes up-to-date.

The two events during the incremental update of the vocabulary that trigger a change in the forward filesystem representations are when (1) a new point is embedded into the vocabulary tree, or (2) any points in the vocabulary tree are released and restructured.

Point Insertion

To insert a new point, we first embed it into the vocabulary tree and follow a parallel path through the corresponding pyramid, creating new bins along the way if necessary. If a new bin is created, we immediately attach pointers from the corresponding nodes in the tree to the new bins.

The running time of this update is the same as the running time of a single point insertion, because the process of updating the bins follows a path down the pyramid that is parallel to P, and at most one new bin is created per node in P.

Re-clustering

If a newly inserted point causes a node in the vocabulary tree to exceed its capacity, thereby forcing local subtree to restructure itself, all of the affected pyramids must be modified accordingly. Let x be the root of the subtree containing all of the restructured nodes, and let Φ be the set of all points that were released (i.e., the union of all of the inverted files at the released leaves). From Φ we can determine the set of all affected pyramids (observe that although using the pointers stored in x for this task may seem more efficient, it would cause us to include more pyramids than necessary, since not all children of x were necessarily released and restructured). Given the set of affected pyramids, and pointers to the relevant subtrees in each of the pyramids (provided by x), performing the update is straightforward: we first delete all of the nodes corresponding to those that were released, and then insert each point in Φ into its respective pyramid, creating and linking new bins as necessary (as above). This is performed quickly by beginning the insertion procedure from x rather than from the root of the tree.

3.3.4 Combining multiple trees

There are two main benefits to combining the results of multiple incrementally-constructed vocabulary trees. First, a forest can reduce quantization effects near the boundaries of nodes in individual trees. Second, we can improve the performance during the early stages of exploration of a new area of the feature space (e.g., a new class of images is introduced). The incremental nature of our method makes it practical to use a set of trees because each tree is updated quickly; approaches that perform batch clustering become prohibitively costly.

For indexing, our goal is to identify the best overall candidate images. Given a query image, each vocabulary tree in the forest recommends a list of candidate images and their matching scores. We merge these lists into a single list. For each candidate on the merged list, we compute the total score by summing the scores from all lists. We rank the candidates according to their total scores and return them as

the retrieval result.

For category learning, we compute a single kernel based on a vocabulary forest. We define the pyramid match forest kernel to be the average of pyramid match kernels for each tree in the forest.

3.4 Experimental Evaluation

To evaluate and validate the approach described in the previous section, five sets of experiments were created to address the following questions:

1. Can several vocabulary trees be combined to improve recognition performance? (Section 3.4.2)

2. Does the performance of a vocabulary tree suffer if it remains static rather than adapting to new data? (Section 3.4.3)

3. How do the growth parameters affect the recognition performance, training time, and the size of the tree? (Section 3.4.4)

4. How does pruning a tree to limit its size affect recognition performance? (Section 3.4.5)

5. How does the performance of an incrementally grown tree compare to one trained using all of the data? (Section 3.4.6)

These questions can be answered in the context of both image indexing and category learning.

3.4.1 Experimental setup

Indexing

There were three datasets used to test indexing tasks: the Zubud dataset [24], the UKY dataset [21], and a new FamousLandmark dataset. The Zubud dataset contains 1,005 images of city buildings in groups of five. The UKY dataset consists of 6,376

images in groups of four, ranging from daily objects to city scenes. The FamousLandmark dataset, which was created by collecting images from a popular online album service (Flickr), consists of 1,000 images of famous landmarks around the world (e.g., Empire State Building, Taj Mahal) in groups of ten.

The indexing tests use the same image representation (128-dimensional SIFT features [17] extracted from maximally stable extremal regions [18]) and scoring methods as [21]. To simulate a dynamic environment, at each time step i, one randomly-selected new image X_i is added to the database by updating the vocabulary tree with X_i's features. The following image X_{i+1} is then used as a query to retrieve matching images from the i images already indexed by the system. A retrieval attempt is considered *accurate* when the top-ranked retrieved image (1-NN) belongs to the same group as X_{i+1} (recall that the datasets consist of groups of matching images). After recording the accuracy, X_{i+1} is inserted into the system and X_{i+2} becomes the next query image. Note that at a given time, there may not be any image from a particular group in the database yet; a retrieval attempt could fail simply because the query image is the first instance of the group seen by the system.

In the context of dynamic real-world problems, such as online news stories, new images are added to the database very frequently and people tend to be interested in finding them. Such scenarios highlight the need to adapt quickly to new data so that recognition accuracy remains high for the current set of images. The adaptive forest method tunes the vocabulary model based on the most recent inputs. Thus, since we are most interested in the performance of our algorithm on test examples that reflect the current trend of data, the performance of this method is compared to those that do not adapt the vocabulary. Therefore, the indexing experiments are evaluated by their *recent accuracy*, which is defined at time i to be

$$A_T(i) = \frac{1}{T} \sum_{j=i-T}^{i} R(j) \tag{3.1}$$

where T is the number of recent retrieval attempts under consideration, and $R(j)$ is a binary value that indicates whether the retrieval attempt on the jth image was

accurate. By considering only retrieval attempts in the recent history, this metric can better reflect the current performance of the system. For the experiments described in this section, $T = 100$ for all cases.

Category Learning

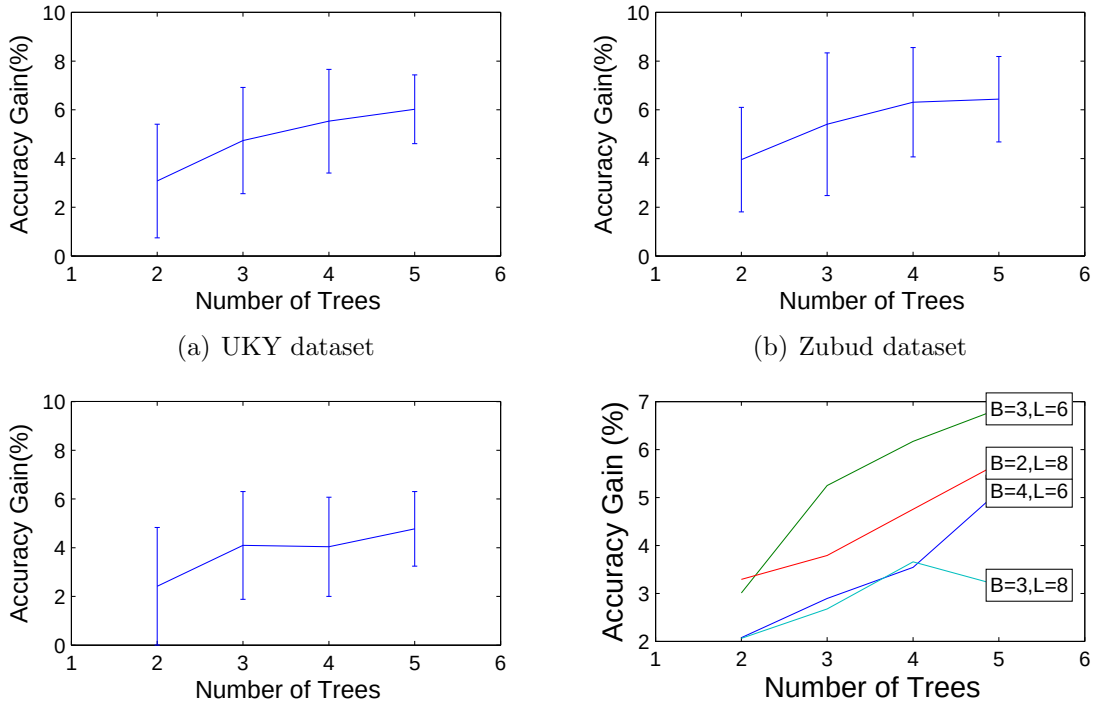
The category learning experiments used the Caltech-101 dataset [9] as a benchmark, which contains 101 classes of object images. Here, a one-vs-all SVM is used as a classifier and the performance is reported by the *mean recognition rate* as in [10]: the score for a single class is the percentage of correctly classified test examples of that class, and the mean recognition rate is the average score over all classes. The image representation is also the same as in [10]: 10-dimensional PCA-SIFT [13] features concatenated with normalized position coordinates, uniformly sampled over 8-pixel intervals. For the tasks reported below, SVM training was relatively fast due to the size of the input and was repeated every time the vocabulary was adapted. However, this method could become prohibitively costly in larger domains, and for those, local SVM recognition methods [30] or incremental SVMs [28] can be used.

3.4.2 Forests vs. trees

The objective of the first set of experiments was to determine whether a vocabulary forest can improve the recognition performance of individual vocabulary trees. Five vocabulary trees were trained using the adaptive method, and the performance of the ensemble was compared to that of a single tree.

Figure 3-3 shows the benefits of forests for indexing (the first three graphs) and for category learning (fourth column). In indexing experiments, larger forests always resulted in better *recent* retrieval accuracy regardless of how many images the system had already indexed and which dataset we tested on. The figures show the *accuracy gain* defined as the percentage difference in accuracy versus a single tree.

The category learning experiments gave similar results. As the size of the forest increases (along the x-axis), the gain in accuracy (compared to the performance of a



(a) UKY dataset

(b) Zubud dataset

(c) FamousLandmark dataset

(d) Caltech-101 dataset

Figure 3-3: **Forests vs. Trees**: The accuracy gain (the absolute increase in accuracy over that of one tree) is greater when more trees are included in the forest. In indexing experiments ((a), (b), and (c)), results are averaged over five simulated runs of dynamic tree creation as images are inserted incrementally. In the category learning experiments (last plot), several tree growth parameters were tested and plotted as separate curves (See Section 3.4.2).

single tree) also increases. Here I report the results over all 101 classes in the dataset but with varying growth parameter settings. Figure 3-3 shows that a forest confers the greatest benefit to shallow trees composed of fewer words; this is useful to our method because the relevant branches will be shallow during early stages of expansion into new parts of the feature space (e.g., when new categories are first encountered).

3.4.3 Adaptive vs. static trees

In the second set of experiments, I studied the impact of adaptation (or lack thereof) on a vocabulary tree's performance in a dynamic environment. For the indexing tasks, I simulated the dynamic nature of the problem by adding images to a vocabulary tree one at a time, and for the category learning task, images were added two classes at a

time. The trees were forced to stop adapting after seeing certain amounts of data to see the effects on performance. The results shown in this section are averaged over five runs of each experiment.

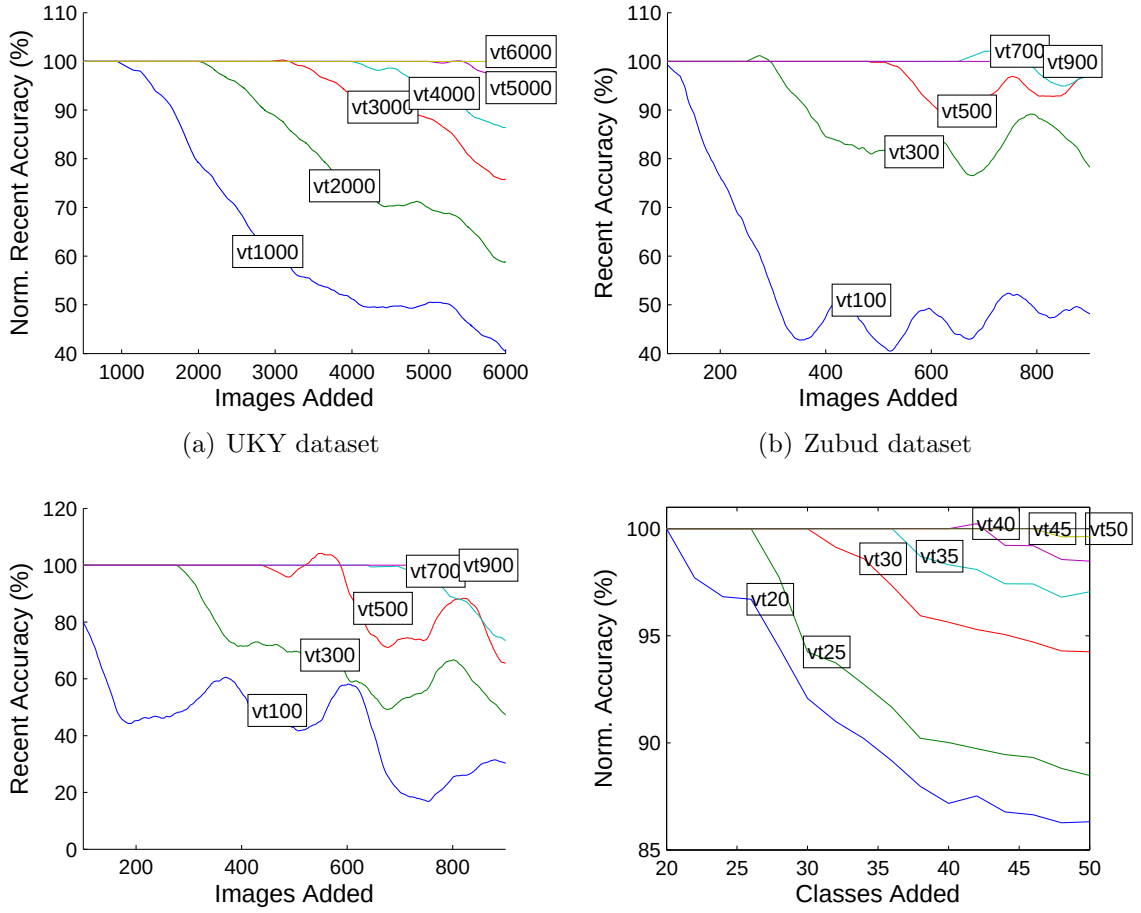
Figure 3-4 shows that for indexing and for category learning, as more images are added to the database (along the x-axis), the performance of the vocabulary tree declines as soon as the vocabulary stops adapting (i.e., where the performance curve branches off from the highest one). These results suggest that vocabulary trees in a dynamic environment should not stop growing; the moment adaptation stops, the ability to discriminate new images begins to deteriorate.

3.4.4 Impact of growth parameters

In this section, I describe the effect of the growth parameters C, S, and R introduced in Section 3.3.1 on the recognition rate, the number of times the tree is restructured, and the size of the tree. Each experiment was done for two different branch factors ($B = 6, 10$), and S was always chosen to be proportional to C so as to keep the number of nodes affected by a re-clustering approximately constant.

Figure 3-5 shows that choosing a larger value of C causes both the size of the tree and the number of times we restructure it to decrease dramatically. Although we would normally expect the recognition rate to decrease as the number of nodes in the tree decreases, we attribute the stability in accuracy to the fact that S was chosen proportional to C, so that although restructuring happened less frequently for higher C, it was performed on more points. This shows that although more visual words in the vocabulary may lead to better performance simply due to more granularity in the feature space, the structure of the tree is also an important factor. This further highlights the need to adaptively update vocabularies: as the types of points being inserted into the tree change, it may be necessary to restructure parts of the tree to ensure a more useful distribution.

The next set of experiments tested the effect of S. We can see that as the size of the neighborhood increases, fewer updates to the tree are necessary with a constant C. This is because after each re-clustering operation, the points in each of those



(a) UKY dataset

(b) Zubud dataset

(c) FamousLandmark dataset

(d) Caltech-101 dataset

Figure 3-4: **Adaptive vs. static trees**: The performance of a vocabulary tree begins to decline when it stops adapting to new data. The label vtN refers to the performance curve of a vocabulary tree that stopped adapting after seeing N images in indexing experiments or N classes in category learning experiments. The curve for a particular vocabulary tree is normalized by the accuracy of that tree at the moment it stopped adapting (see Section 3.4.3).

nodes become better distributed (based to R, which we have held constant). Thus, as expected, size of the tree increases as S is increased. Observe that the recognition rate was not sensitive to the choice of S. This is because S has two competing effects on the structure of the tree: a higher value means that each re-clustering operation includes more nodes, causing larger substructures of the tree to be updated; at the same time, because the capacity constraint is triggered less often due to better overall point distribution after each re-clustering, the restructuring happens less often. Again, this suggests that the number of visual words present in the tree does not play as significant a role as the structure of the tree itself.

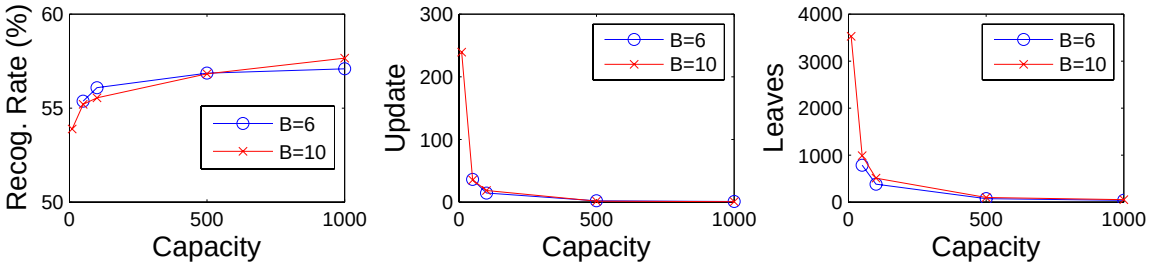
Figure 3-5 shows that increasing R has the opposite effect as increasing S: with a high R, the criterion for redistribution is weak, leaving the resulting nodes possibly still very close to their capacities. This, in turn, causes updates to happen more frequently because the capacity constraint is triggered more often. As a result, the number of leaves in the tree becomes smaller but each becomes more populated.

Because varying the parameters does not significantly impact the recognition rate, they should be chosen according to the particular usage scenario (i.e., whether speed or size is favored). Having a smaller representation is not detrimental to performance because the structure of the vocabulary tree is changed as needed. However, such a representation is achieved at the cost of a higher number of updates to the structure.

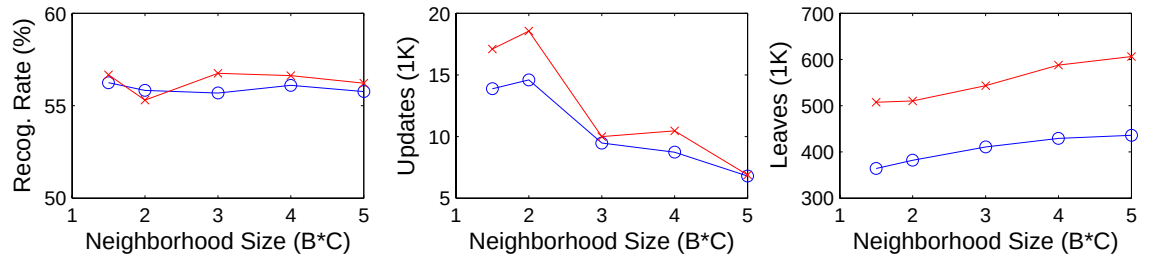
3.4.5 Pruning

To study the effect of pruning on the resulting vocabulary, I simulated a dynamic category recognition scenario on the Caltech-101 database. After the vocabulary tree was shown a set of images from a given class, the recognition performance was tested on the 10 most recent classes it had previously seen, including the newly added one. Reported here is the recognition performance averaged over 10 SVM runs with 15 training examples per class as well as the size of the tree in terms of the total number of leaves.

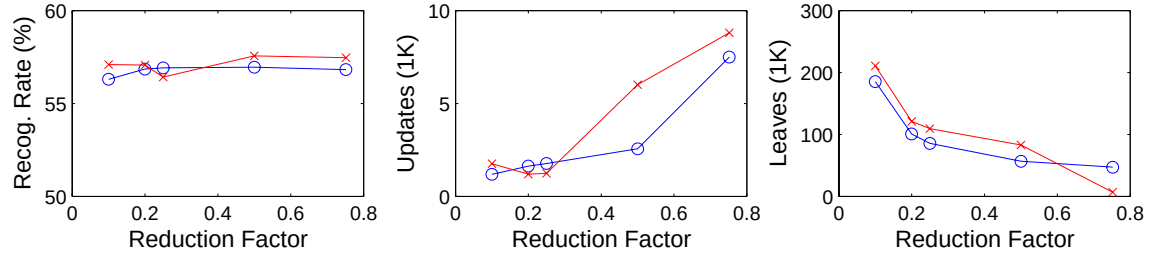
As expected, the total number of leaves in the tree (shown in Figure 3-6) remained constant for a pruned tree, while the number of leaves in the unpruned tree grew



(a) Varying Capacity (C) $\left(R = \frac{1}{3}; S = 2BC\right)$



(b) Varying Neighborhood Size (S) $\left(C = 100; R = \frac{1}{3}\right)$

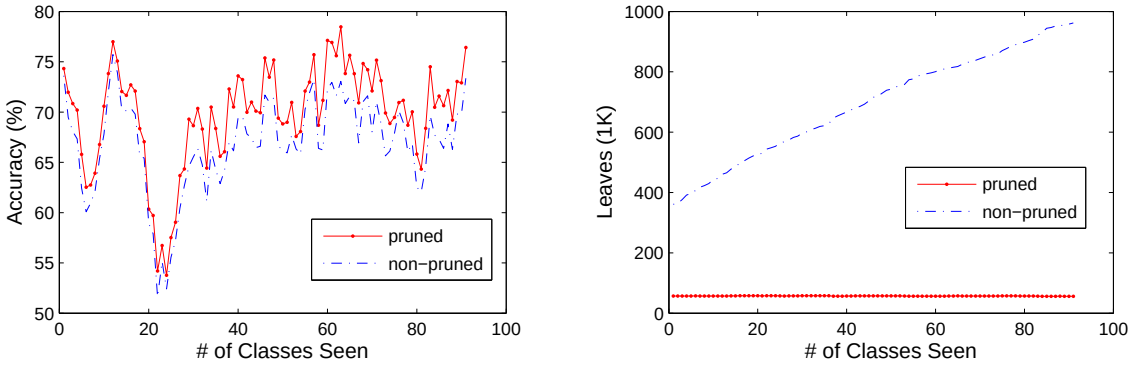


(c) Varying Reduction Factor (R) ($C = 500; S = 10BC$)

Figure 3-5: **Impact of growth parameters**: the recognition performance is insensitive to the parameters, but the size and speed can vary dramatically (see Section 3.4.4.) and thus should be chosen based on the particular application.

linearly as new images were added. We also observe that the recognition rate for the most recently seen classes was always higher for the pruned tree than the unpruned tree, despite the limit on the number of visual words.

Pruning is beneficial to the performance of an incrementally grown tree because it allows for the restructuring of heavily populated areas of the vocabulary tree. Recall that the restructuring algorithm chose to ignore nodes which contained too many points ("unreleasable" nodes) to avoid violating the neighborhood size constraint (S). As a result, structural improvements to those areas of the vocabulary tree cannot happen unless those nodes become obsolete and are pruned. Once pruned,



(a) Accuracy (b) Size

Figure 3-6: **Pruned vs. Non-pruned Trees**: Pruned trees achieve better recognition accuracy while maintaining a constant size (see Section 3.4.5).

that area can be repopulated with data from new categories, which may prefer a different distribution of the bins in that part of the feature space. As a result, the structure of the important parts of the vocabulary tree are kept up-to-date to reflect the most recently seen data. This finding further supports our claim that the structure of the tree is an important factor in determining the quality of the vocabulary; since different categories may prefer different distributions of the visual words in the same parts of the feature space, it is important to adapt the tree's structure to newly seen images.

3.4.6 Incremental vs. global clustering

The objective of the final experiment was to demonstrate that trees incrementally grown would be comparable to the trees constructed from batch computation on the entire dataset. To this end, I created ten sets of problems with a varying number of Caltech101 classes, applying both incremental and the baseline global strategies to train vocabulary trees and compare their recognition performances. Also, for simplicity, when forming the vocabulary, both incremental and baseline global methods have access to all the images in the given subset of images.

Figure 3-7 shows the adaptive trees trained incrementally rivals the performance of a tree built using the entire dataset. Note that regardless of the size of the problem (x-

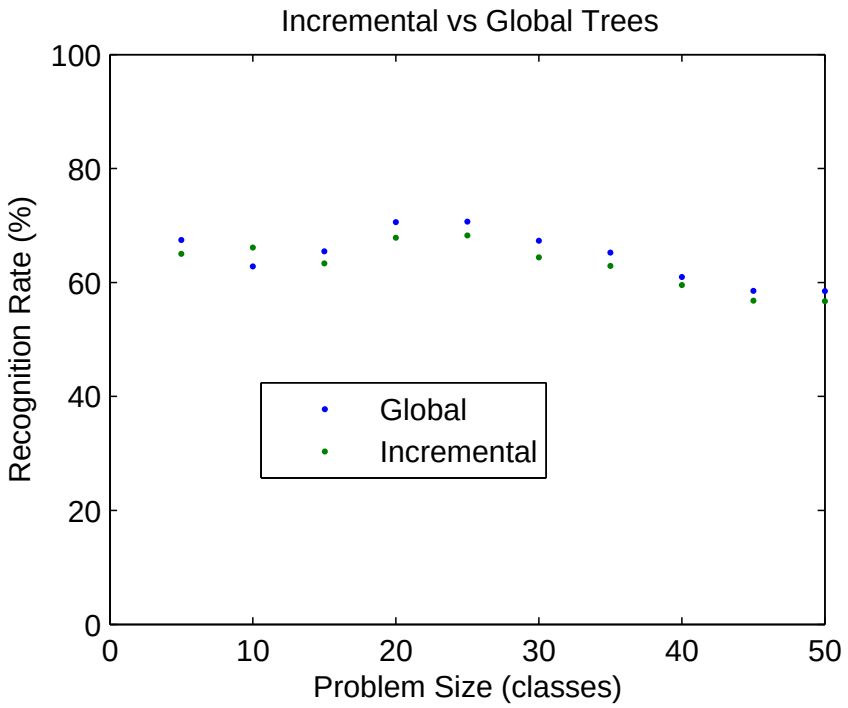


Figure 3-7: **Incermental vs. Global Clustering**: Incremental trees provide the same level of performance as trees retrained globally from all the data.

axis), the incremental trees can always offer the same level of recognition performance (y-axis) as ones constructed globally from the entire dataset. These results show that trees grown incrementally are comparable to trees obtained via batch computation, but are considerably faster and suitable for causal and dynamic recognition tasks.

3.5 Discussion

Adaptive vocabulary forests are suitable for large-scale recognition and retrieval tasks because they can operate efficiently in a dynamic environment. A forest of vocabulary trees outperforms a single tree; this can be easily implemented by having multiple, independent trees can be distributed across a number of servers. Such a system is naturally resilient to failures; if one server becomes unavailable, the ability of the system to function is unaffected and can still perform the same tasks without any modifications. As increasing the size of the forest provides diminishing returns, operating a forest with a critical mass of servers results in very small (if any) performance

drops if one of the servers stops functioning.

The results presented in Section 3.4 shows that adaptive trees are not only faster, but also achieve the same level of accuracy that the slower, batch methods do. Pruning trees allows us to constrain the maximum size of the trees; if the application is sensitive only to recent data, pruned trees offer a much more compact representation and can also boost recognition accuracy. Because the recognition accuracy is insensitive to the choice of the growth parameters, implementers can easily choose to make time-space tradeoffs by tweaking parameters appropriately. This versatility makes adaptive vocabulary forests suitable for a wide range of applications.

Algorithm 4 Given a set of points Φ released from the vocabulary tree, the set of affected pyramids $\mathcal{P}_\Phi$, and the root x of the restructured subtree, update the image pyramids

```
procedure UpdatePyramids(Φ, P_Φ, x)
    for all Δ ∈ P_Φ do
        RemoveAffectedBins(Δ, x)
    end for
    for all p ∈ Φ do
        Δ_p ← pyramid of p's source image
        InsertIntoPyramid(Δ_p, p)
    end for
end procedure

procedure InsertIntoPyramid(Δ, p)
    w ← p's matching visual word
    path ← path in the vocabulary tree to reach w
    for all x ∈ path do
        x_Δ ← x's pointer into Δ, if it exists
        if x_Δ does not exist then
            Create a new x_Δ and add it to Δ
            Link x_Δ to x with a pointer
        end if
        Add p to x_Δ
    end for
end procedure

procedure RemoveAffectedBins(Δ, x)
    x_Δ ← the bin in Δ pointed to by x
    for all c_Δ ∈ children(x_Δ) do
        c ← the node in the vocabulary pointed to by c_Δ
        if c was removed by ReleasePoints then
            remove(c_Δ)
        end if
    end for
end procedure
```

Chapter 4

VT-SVM: Fast SVM with Vocabulary Trees

4.1 Overview

Object recognition systems designed for Internet applications typically need to adapt to users' needs in a flexible fashion and scale up to very large data sets. In this chapter, I analyze the complexity of several multi-class SVM-based algorithms and highlight the computational bottleneck they suffer at test time: comparing the input image to every training image. I propose an algorithm called *VT-SVM* that overcomes this bottleneck; it offers not only the efficiency of a simple nearest-neighbor classifier, by voting on class labels based on the k nearest neighbors quickly determined by a vocabulary tree, but also the recognition accuracy comparable to that of a complex SVM classifier built offline with a vocabulary-tree-based kernel, by incorporating SVM parameters into the voting scores incrementally accumulated from individual image features. Empirical results demonstrate that adjusting votes by relevant support vector weights can improve the recognition accuracy of a nearest-neighbor classifier without sacrificing speed. Compared to existing methods, this algorithm achieves a ten-fold speed increase while incurring an acceptable accuracy loss that can be easily offset by showing about two more labels to the user in the result. The speed, scalability, and adaptability of VT-SVM makes it suitable for large-scale Internet

vision applications.

4.1.1 Time complexity of current systems

Many object recognition systems have similar processing pipelines. In this section, I examine the time complexity of some recent object recognition methods and describe how their performance is evaluated. As many of them are based on hierarchical bag-of-words representations, they have many steps in common.

The image processing pipeline for many modern object recognition systems is as follows:

1. Extract features from all training images
2. Preprocess and quantize the features
3. Build an intermediate representation of each image (e.g., a bag of words)
4. Compute the similarity between each pair of training images
5. Train an SVM with the above kernel matrix

The preprocessing step is needed for methods like [11] and [14], where the feature points are clustered into some number of visual words (this clustering may be done hierarchically as well). The following step, in which intermediate representations are created, could include building feature histograms and/or pyramids.

Classifying a new image involves the following steps:

1. Extract features from the new input image I
2. Convert I into an intermediate representation
3. Compute the similarity between I and each support vector training image

These systems are typically evaluated by their performance on benchmark data sets, such as Caltech-101 [9]. In these particular benchmark tests, the standard procedure is to train with a fixed number of examples per class and report the mean

	SVM-KNN [30]	VGPMK [11]	Spatial PMK [14]	PHOG [3]	This work
Preprocess (1 image)	none	$O(fbdL)$	$O(fdw)$	$O(f)$	none
# comparisons per query	$O(k^2)$	$O(n)$	$O(n)$	$O(n)$	$O(k)$
Comparison time (1 pair)	$O(df^2)$	$O(f)$	$O(fw)$	$O(fL)$	$O(df)$
Total comparison time	$O(k^2df^2)$	$O(nf)$	$O(nfw)$	$O(nfL)$	$O(kdf)$

Table 4.1: Query time complexity of selected object recognition systems (Section 4.1.1). n is the number of training images, f is the number of features extracted from an image, and d is the feature dimension. For tree-based methods, b is the branch factor and L is the number of levels. w is the number of visual words in the spatial PMK method. k is the size of the local neighborhood in SVM-KNN and the proposed method.

recognition rate, where the system predicts one label for each test image and the accuracy is normalized per-class by the number of test examples for that class.

The test-time complexity for several algorithms known to perform well on Caltech-101 is given in Table 4.1. These methods are described in detail in Section 2.3. The best known performance involves combinations of these methods [26, 3], but they are analyzed separately here; the combination methods take at least as much time as the slowest component.

Observe that at test time, the major scalability issue is step 3: comparing the query image to every support vector in the database. As the number of object categories grows, these systems become less and less able to support large numbers of simultaneous queries. In the following section, I describe a novel method for quickly approximating SVM decision values without suffering this bottleneck.

4.2 Motivation

The vocabulary tree technique proposed by [21] is an efficient algorithm for retrieving the r images that are the most visually similar to an input query image I. Each leaf node in a vocabulary tree corresponds to a visual word. Stored in the leaf node is an inverted index holding pointers to database images that contain the corresponding visual word. This indexing scheme was first developed for document retrieval and has been successfully extended to image retrieval.

To apply a vocabulary tree to category-level problems such as object recognition, the most straight-forward approach is to vote on the k nearest neighbors retrieved by the tree. In its original format designed for image retrieval, a vocabulary tree allows each feature f in an input image to cast a vote for each image in the set training images referenced by the inverted file that f corresponds to. In the proposed format extended to object recognition, a vocabulary tree permits votes to be cast by f directly on the class labels associated with these training images.

However, the voting strategy described above has limitations. Each feature f is only permitted to cast *positive* votes on any of the class labels. Sometimes, it may be desirable to allow f to also cast *negative* votes, especially on certain classes for which the training images found to also contain f are known to be strong negative indicators for these classes. For example, a training image of the class *dog* may turn out to be a particularly strong negative indicator for the class *car*. Therefore, a feature indicative of a dog found in the input image I should be allowed to cast not only a positive vote on the class label *dog*, but also a negative vote on the class label *car*.

SVM learning is an effective way of determining which subsets of the training examples are strong negative indicators of a given class label. These relevant negative examples manifest themselves as negative support vectors, lying on the boundaries of decision hyperplanes learned to separate positive examples from negative examples of a particular class. These negative support vectors contain useful knowledge that can help us improve the recognition accuracy of a vocabulary tree-based nearest neighbor classifier, by factoring in the weights of these support vectors in vote calculation.

However, classifying an input image I by evaluating a multi-class SVM requires the knowledge of the similarity between the input image I and *every* image in the training set that acts as a support vector. In practice, with the Caltech-101 data set, nearly every training image becomes a support vector in at least one of the 101 SVMs. Thus, full evaluation of the multi-class SVM can be computationally expensive because of the need to exhaustively compare the input image to every image in the training set.

In summary, while a vocabulary tree has a speed advantage, it suffers in accuracy

Algorithm 5 Train a vocabulary tree and a multi-class SVM (Section 4.3.2) and use them together to classify an input image i (Section 4.3.3)

```
procedure TRAIN
    Train a vocabulary tree
    for all c ∈ known class labels do
        Train a 1-v-all SVM, storing α_{i,c} and b_c
    end for
end procedure

function CLASSIFY(i)
    F ← EXTRACTFEATURES(i)
    for all c ∈ known class labels do
        s_c ← b_c
        for all f ∈ F do
            w ← GETVISUALWORD(f)
            T_w ← GETINVERTEDFILE(w)
            for all t ∈ T_w do
                s_c ← s_c + s(f, w)α_{t,c}
            end for
        end for
    end for
    return mapping of c → s_c, sorted by s_c
end function
```

because it is unable to cast negative votes. On the other hand, a multi-class SVM classifier has superior accuracy but runs slower because it compares an input image to every training image. This algorithm explores how these methods can work together to complement each other's weaknesses with their respective strengths.

4.3 The VT-SVM Algorithm

In this section, I describe the VT-SVM (vocabulary tree SVM), an efficient object recognition algorithm ideal for Internet applications, which offers the speed of a vocabulary tree-based nearest neighbor classifier and the accuracy of a multi-class SVM classifier. Below, I motivate the combination of SVMs and vocabulary trees (Section 4.2), present the approach (Section 4.3.1), describe the details of the training and testing algorithms (Sections 4.3.2 and 4.3.3), and discuss its efficiency (Section 4.3.4), adaptability, and scalability (Section 4.3.5).

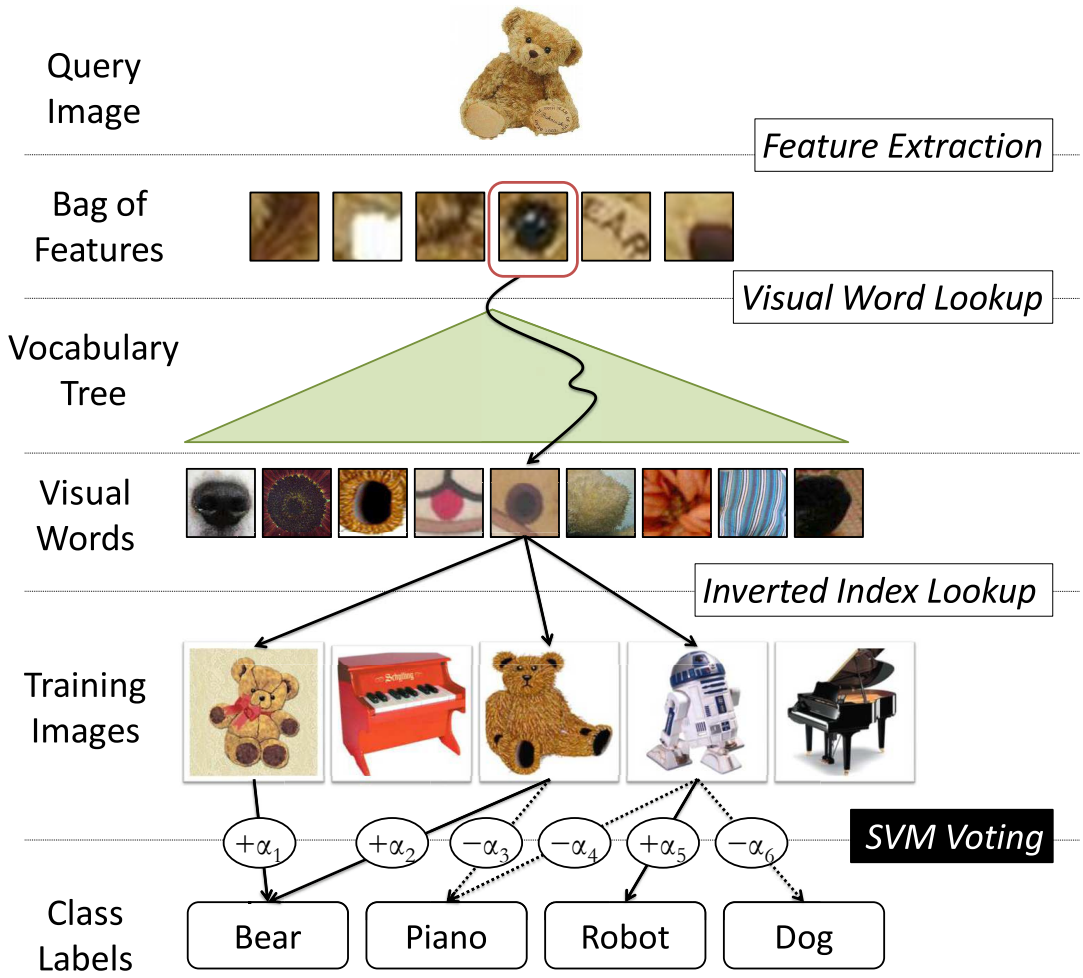


Figure 4-1: Classifying an image using nearest-neighbor voting based on a vocabulary tree. Both positive and negative support vector weights (α) are factored in the voting scores to improve the recognition performance (Section 4.3.3)

4.3.1 Overview of the algorithm

VT-SVM seeks to offer not only the efficiency of a vocabulary tree but also classification performance comparable to that of a multi-class SVM classifier. To achieve efficiency, this method depends on a vocabulary tree-based nearest neighbor classifier to quickly cast votes on class labels. To achieve accuracy, this method weights these votes based on relevant SVM parameters—in particular, the support vector weights and offsets. In essence, this method takes advantage of the synergy between a vocabulary tree and an SVM. On the one hand, it avoids an exhaustive scan of the database by adopting a voting strategy on the nearest neighbors quickly identified by

a vocabulary tree. On the other hand, it approximates the behavior of a multi-class SVM by incorporating its parameters in vote calculation, without incurring the high computational cost of full SVM evaluation.

Section 4.3.2 describes how to train the model by constructing a vocabulary tree and deriving SVM parameters and Section 4.3.3 explains how to classify new query images by using the vocabulary tree to cast votes on class labels based on individual features and weighting these votes using the SVM parameters to improve recognition performance. The entire algorithm is summarized in Algorithm 5.

Unlike the SVM-KNN method of [30], this approach does not create new classifiers online. While SVM-KNN finds the nearest neighbors of a query image and trains a new multi-class SVM using these nearest neighbors online, this method trains the SVM offline and uses the learned parameters to weight the voting scores based on the nearest neighbors online. SVM-KNN performs its largest computations online for each query image, whereas this method keeps the amount of online computation to a minimum.

4.3.2 Building a VT-SVM offline

At training time, two tasks need to be performed. The first task is to construct a vocabulary tree following the procedure described in [21] and populate it with the training images to build an inverted index. The second task is to learn the SVM parameters that can be used to improve a vocabulary tree-based nearest neighbor classifier. To construct a kernel matrix, we use the bag-of-words histogram method as described in [27], where the similarity between two images is given by the intersection of the normalized visual-word histograms representing each image:

$$K(i,j) = \sum_{w \in W} \min(H_i(w), H_j(w)) \tag{4.1}$$

Based on this matrix we apply the standard multi-class 1-vs-all SVM training procedure to build an ensemble of SVMs, one for each class. Specifically, we record each $\alpha_{i,j}$—the support vector weight of the i-th training image in the SVM for the

j-th class, and each b_j— the offset of the SVM for the j-th class.

It is important to note that the SVM must be trained with a kernel matrix constructed with the vocabulary tree similarity scores, so that the retrieved nearest neighbors in the classification step correspond to training examples that are likely to lie close to an SVM decision boundary. The need for this property becomes clear at test time, which is described in the following section.

4.3.3 Classifying input images online with VT-SVM

At query time, the algorithm uses the vocabulary tree and SVM parameters derived in the training stage to quickly and accurately recognize an input image. Figure 4-1 provides an overview of the method. An input image I is first converted to a bag-of-features representation, denoted as $F : \{f_1 \dots f_p\}$, where fs are the features. Let V_j be the total number of votes the class j receives from all the features in F. The goal is to incrementally accumulate the partial vote $\Delta V_j(f)$ contributed by each individual feature $f \in F$. As we iterate through all the features in F, the total vote for a class label j is

$$V_j(I) = \sum_{f \in F} \Delta V_j(f) \tag{4.2}$$

Let us consider how to compute $\Delta V_j(f)$. First, we look up the visual word w_f for the feature f in the vocabulary tree. Then, from the inverted file stored in w_f, we obtain a list of relevant training images $T_f : \{t_1 \dots t_q\}$. Each training image $t \in T_f$ can play one of the following three roles with respect to a particular class j: a positive support vector if $\alpha_{t,j} > 0$, a negative support if when $\alpha_{t,j} < 0$, of a non-factor if $\alpha_{t,j} = 0$, where $\alpha_{t,j}$ is the support vector weight of the training image t for the class j.

The partial vote $\Delta V_j(f)$ reflects the influences of two factors: (1) $s(f, w_f)$: the similarity between the feature f and its corresponding visual word w, and (2) $\alpha_{t,j}$: the support vector weight assigned to the training image t in the SVM for the class j. Thus, $\Delta V_j(f)$ can be computed as follows::

$$\Delta V_j(f) = \sum_{t \in T_f} s(f, w_f)\alpha_{t,j} \tag{4.3}$$

.

Combining Equations 4.2 and 4.3, we obtain the weighted votes $V_j(I)$ for each class j. The class label predicted by this method simply finds the highest such value:

$$\arg\max_j V_j(I) = \sum_{f \in F} \sum_{t \in T_f} s(f, w_f)\alpha_{t,j} + b_j \tag{4.4}$$

where b_j is the SVM offset for class j.

It is useful to compare the above formula to a conventional voting strategy that considers no negative votes and receives no assistance from SVM parameters, expressed as:

$$\arg\max_j V_j(I) = \sum_{f \in F} \sum_{t \in T_f \cap T_j} s(f, w_f) \tag{4.5}$$

where $T_f \cap T_j$ denotes the set of training images that carries the class label j and are referenced by the inverted file related to the feature f; these are the images responsible for positive votes. Since $s(f, w_f)$ does not depend on t, (4) can be further simplified to:

$$\arg\max_j V_j(I) = \sum_{f \in F} |T_f \cap T_j| s(f, w_f) \tag{4.6}$$

where $|T_f \cap T_j|$ denotes the number of training examples that shares a feature f with the input image.

Because we choose only a fixed number of support vectors (the ones that are highest-ranked by the vocabulary tree), it is crucial that those with the highest magnitudes of weights that are relevant to the given query image appear within the top k examples. In Section 4.4 I show empirically that the recognition performance of our method is superior to that of a conventional voting strategy (Section 4.4.1) and is comparable to that of a multi-class SVM classifier built using the same kernel (Section 4.4.3). Additionally, I compare the performance of VT-SVM to other well-known

object recognition algorithms in the context of both standard SVMs and the nearest-neighbor SVM-voting strategy (Section 4.4.4), showing that the latter approach works best when a vocabulary tree is used to generate the kernel.

4.3.4 Computational efficiency

The efficiency of this approach comes from three sources. First, the inverted file stored in the vocabulary tree makes it fast to look up nearest neighbors by focusing on those training images that actually share similar features with the input image, allowing us to skip many training images that bear no resemblance to the input image.

Second, to speed up the process of vote calculation, we can adopt a greedy approach that uses only the k nearest neighbors, which are the training images most similar to the input image. As the algorithm iterates through each feature in the bag-of-features representation of the input image and incrementally accumulates the votes each feature casts on the training images receive, we can set the limit on the size of the candidate list. This is the same heuristic that has been tested successfully in [21].

Lastly, in addition to being greedy on the top training images, we can extend the greediness to the top r classes as well. After having processed a sufficient percentage of the features in the input image, for example, 25%, we can start focusing exclusively on the first r vote-receivers among the class labels, because the final winner is more likely to come from these forerunners.

4.3.5 Adaptability and scalability

Adaptability refers to a system's ability to update itself to react to the changes in training images or class labels. To add or remove training images, this method can use the adaptive algorithm described in Chapter 3 to incrementally update the structure of the vocabulary tree. To add or remove class labels, we can use the incremental algorithm described in [4] to update the SVM efficiently.

Scalability refers to a system's ability to handle a large volume of user traffic and

image data. For SVM parameters, since they are trained offline, they have little effect on the scalability of the online application. For the vocabulary tree, to improve the throughput for serving user requests, we can split the tree into parts that can be hosted on separate servers in a manner analogous to term-partitioning. To increase the capacity for image data, we can add more branches or levels to the vocabulary tree. As pointed out in Table 4.1, the complexity of this method does not increase as the number of training images n grows, since it depends only on the number of features f and the choice of k.

4.4 Experimental Evaluation

In this section, I describe four experiments, each designed to address one of the following research questions:

1. Does this algorithm improve vocabulary tree-based nearest neighbor voting?

2. Does this algorithm operate effectively with a small number of nearest neighbors?

3. Does this algorithm achieve recognition accuracy comparable to that of an SVM classifier?

4. Does this algorithm provide reasonable time-accuracy tradeoffs with respect to some of the current methods, such as [14, 3, 11]?

All experiments were performed on the Caltech-101 dataset [9]. The training set consisted of 15 examples randomly selected from each of the 101 classes, for a total of 1515 training images. As in [10], each image was represented as a bag-of-features, computed by dense sampling over an uniform grid with 8-pixel spacing. Each sampled point was described by a 12-dimensional descriptor obtained by concatenating a 10-dimensional PCA-SIFT [13] descriptor with normalized x-y coordinates (our approach is equally applicable to other feature detectors and descriptors). The test set was

similarly constructed by choosing 15 instances per class not previously included in the training set.

The recognition performance is expressed as *recall-at-r*, defined as the rate at which an algorithm is able to place a correct item (e.g., URL, label) on the first "page" of r results. In object recognition, for example, a recall-at-5 of 0.65 means there is 65% chance a correct label would appear within top 5 entries of the result. This is a metric commonly used by the information-retrieval community to evaluate Internet search engines. Typical experiments with the Caltech-101 dataset involve the mean recognition rate, which only measures the recall-at-1 performance. The recall-at-1 measure is less relevant for the applications considered here because the user may want to see a ranked list of results, not just the top match.

4.4.1 Effects of SVM weighting on vocabulary trees

The algorithm described in Section 4.3 seeks to improve a vocabulary tree's ability to recognize object categories by using SVM parameters to weight the votes cast by its visual words. The first experiment was intended to verify whether such a weighting scheme could indeed improve the recognition performance of a vocabulary tree-based nearest neighbor classifier. The experiment used $k = 300$ and classified the images in the test set using only the k nearest neighbors of each test image. The output (see Equation 4.4) was compared to a standard nearest-neighbor voting scheme (Equation 4.6) by observing the recall-at-r achieved by each method, where r ranged from 1 to 20 out of 101 total classes.

Figure 4-2 shows that when we weight the votes based on SVM parameters, the performance of a vocabulary tree improves significantly compared to the baseline conventional nearest neighbor-voting strategy, as made evident by the upward shift of the curve. For instance, the recall-at-5 increased from 0.48 to 0.62 using SVM-weighted method. This demonstrates the benefit of incorporating SVM parameters into a vocabulary tree for object recognition.

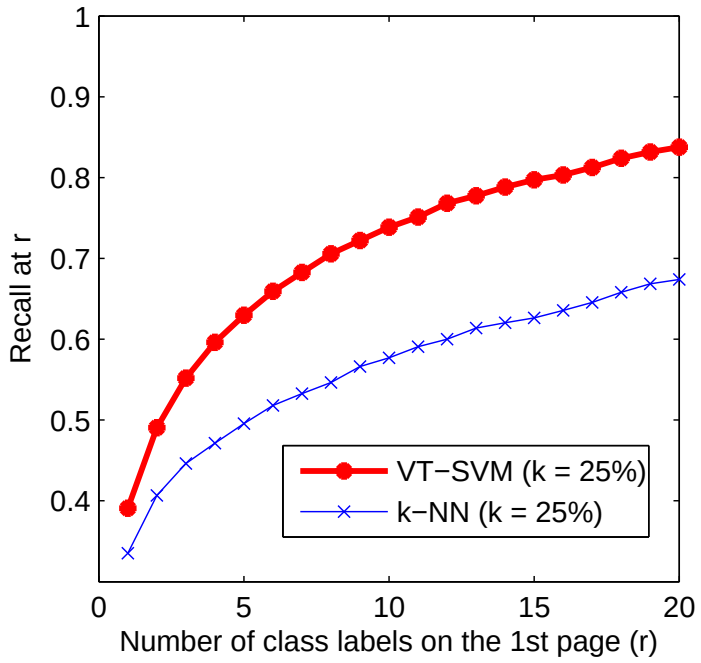


Figure 4-2: The SVM-weighted vocabulary tree voting outperforms the conventional nearest-neighbor voting strategy (Section 4.4.1).

4.4.2 Speed-accuracy tradeoffs

The VT-SVM improves recognition speed by voting with only a small number of nearest neighbors that can be quickly identified using a vocabulary tree, without losing much in recognition performance. The purpose of the next set of experiments was to investigate the speed-accuracy tradeoffs related to the choice of k. It is expected that as k decreases, recognition becomes faster (because fewer training examples are compared to the query), but at the cost of reduced recognition accuracy. In these experiments, k varied between 1515 down to 1. For each k (shown as a percentage of the total number of training examples), the recall-at-r was measured for $r = 1, 3, 5, 10$ and 15.

Figure 4-3(a) shows that as fewer nearest neighbors were considered (x-axis), recognition performance (y-axis) declined. However, the decline was rather slow, until k, the number of nearest neighbors, had fallen below 200, where the decline began to become more drastic. This observation suggests the feasibility of a greedy approach that sets k to about 15% of the training images. The result is a nearly

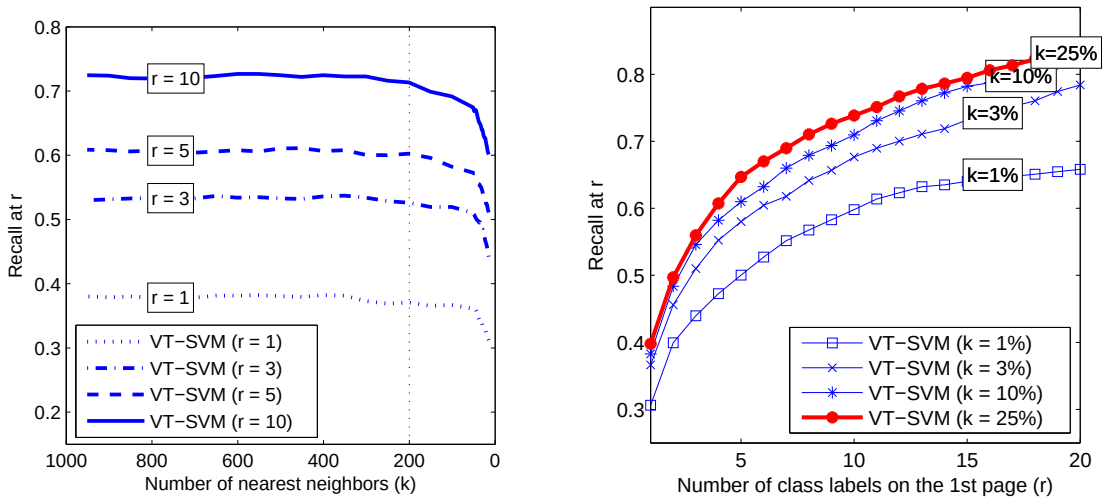


Figure 4-3: Speed-accuracy tradeoffs of VT-SVM (Section 4.4.2).

ten-fold increase in speed without any significant losses in accuracy.

Figure 4-3(b) shows the results from a different perspective. The figure plots the recall-at-r performance with $k = 1\%$, 3%, 10%, and 25% of the training images. As k decreases, the performance drops as indicated by the downward shifts of the curves. Yet, the drop only becomes significant when k is below 10% of the training images.

4.4.3 Comparison to SVM

The VT-SVM algorithm allows a vocabulary tree to leverage the information embedded in SVM parameters to approximate the behavior of an SVM classifier. The purpose of this set of experiments was to measure the recognition performance of the VT-SVM with $k = 25\%$ and compare it to that of a standalone multi-class SVM. Figure 4-4 shows that this method closely approximates the performance of the multi-class SVM classifier. At $k = 100\%$, the two performance curves become identical by definition, but the figure shows that there is almost no loss by setting k to a small fraction of the training examples.

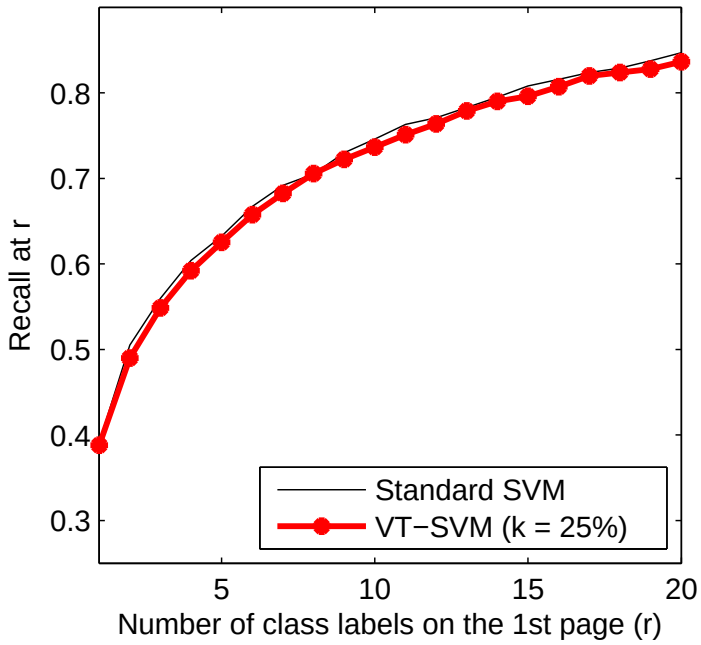
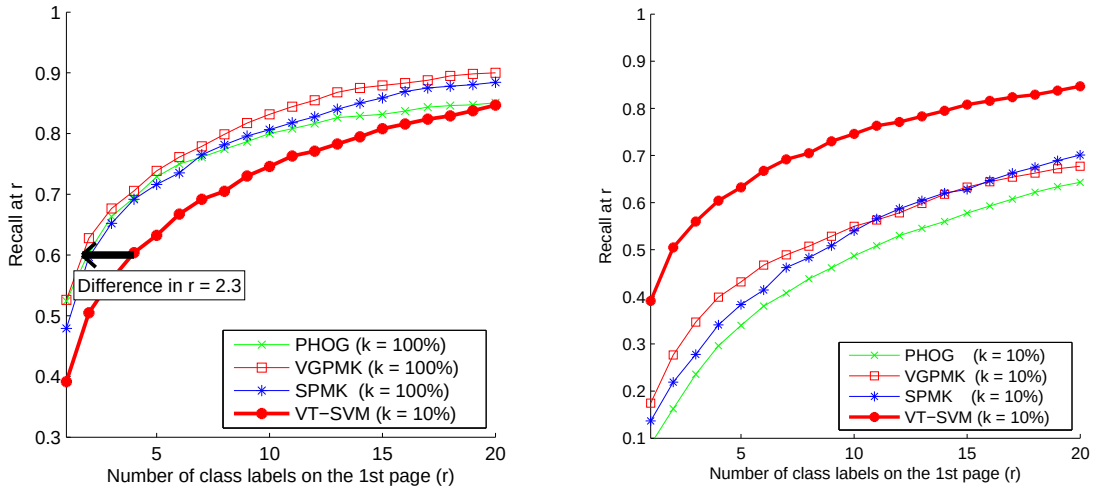


Figure 4-4: Performance comparison to SVM (Section 4.4.3).

4.4.4 Comparison to existing algorithms

This section provides a comparison between VT-SVM and several state-of-the-art algorithms for object recognition. As pointed out earlier, while these algorithms may achieve better recognition performance, they do so at a much higher computational cost. For online applications, users might be less willing to wait for long time just to get a precise, rank-1 result; instead, they may prefer quicker but possibly less precise responses with more items shown on the first page. There are two parts of this experiment: first, the VT-SVM with $k = 10\%$ is compared to three other object recognition algorithms in their normal settings. Then, it is compared to those same three algorithms where only 10% of the nearest neighbors are available to them. The three algorithms used as baselines were the vocabulary-guided pyramid match (Section 2.2.2), spatial pyramid match (Section 2.3.1), and pyramids of orientation histograms (Section 2.3.2).

The first comparison, shown in Figure 4-5(a), demonstrates that the performance of the VT-SVM method is indeed inferior to those of the other methods when the



(a) (b)

Figure 4-5: Performance comparison to other methods (Section 4.4.4).

entire training set is made available to them. However, the performance gap narrows as r increases. More importantly, the result suggests that by showing on average 2.3 more labels in the result VT-SVM performs just as well as the other methods. This is done in exchange for a ten-fold increase in query time, as the VT-SVM only used 10% of the training set.

Next, I consider the accuracy of the other methods when they are forced to use only 10% of the training set– this forces those algorithms to make the same speed-accuracy tradeoff as VT-SVM does. For each test image, the top 10% of the training images with the highest similarities values to the query were chosen and were allowed to participate in the evaluation of the multi-class SVM classifier. Note that in order to identify the top 10% nearest neighbors, these methods might need to compare the test image to every image in the training set, thus giving no actual speed advantage, but for the purposes of comparison, an oracle was present that reported the k nearest neighbors without an exhaustive search. Figure 4-5(b) shows that VT-SVM outperforms existing methods operating under (at best) the same time constraints. This observation suggests that existing methods, while highly tuned to provide optimal rank-1 results, might be less robust to incomplete knowledge of the relationship between a test image and the training images.

4.5 Discussion

In this chapter, I analyzed the time complexity of existing state-of-the-art recognition methods, showing that large-scale applications of these suffer from lengthy query times. I then proposed a new object recognition method called VT-SVM, designed specifically to provide speed, adaptability, and scalability by combining the efficiency of a vocabulary tree with the accuracy of an SVM. The VT-SVM algorithm can achieve a ten-fold speed increase while maintaining comparable accuracy under the recall-at-r metric. In addition, the vocabulary tree used in this algorithm can be made adaptive using the algorithm from Chapter 3 and is easily distributed across many machines, making it suitable for large-scale Internet object recognition tasks.

Chapter 5

Practical Vision Applications With LIBPMK

LIBPMK is a fast implementation of the pyramid match algorithm [10, 11]. This toolkit provides a flexible framework with which developers can quickly match sets of image features and run indexing or classification experiments. LIBPMK provides functionality for k-means and hierarchical clustering, dealing with data sets too large to fit in memory, building multi-resolution histograms, quickly performing pyramid matches, and training and testing support vector machines (SVMs). In this chapter, I describe the design philosophy of LIBPMK (Section 5.1), give details about particular modules in LIBPMK and how they work together (Section 5.2), provide an experimental evaluation of LIBPMK (Section 5.3), and describe some of the practical applications that it has been used for (Section 5.4).

5.1 Design Considerations

The primary goal of LIBPMK is to provide a *flexible* framework with which developers can create object recognition or content-based retrieval applications based on the pyramid match. Because of its versatility, the uses of LIBPMK are not limited only to pyramid match computations; LIBPMK was designed with modularity in mind so that it is easy not only to create and test variations of pyramid matching, but also

other computer vision tasks. A common framework allows for faster development and better comparisons between algorithms.

The second design consideration was for *space-efficiency*. Because image data is generally large, it is generally more useful to be able to fit more in memory, especially for interactive image retrieval applications where quick memory lookups are necessary. Because of this, LIBPMK uses as compact representations as possible to avoid unnecessarily swapping data from disk too often. Because of hardware constraints, small factors (e.g., 2 or 3) in memory usage are extremely important. To ease the burden on developers, LIBPMK provides functions that perform meticulous accounting of memory usage.

I have designed LIBPMK with flexibility and space-efficiency as the highest priorities to facilitate its adoption in the research community. Because researchers are generally interested in building and testing prototypes of new ideas quickly rather than considering the end-user experience with the system, the biggest desire is to have short development times to see whether further development is worth pursuing. However, although LIBPMK was designed with less emphasis on speed, it still operates very quickly; Section 5.3 shows the behavior of LIBPMK on image data in terms of both speed and space usage.

5.2 Components of LIBPMK

5.2.1 Data representation

Image data

LIBPMK provides an efficient representation of the bag-of-features model, where a single image is represented as a set of d-dimensional feature points (a `PointSet`). A list of these, or a `PointSetList`, forms an image database. Feature extraction software produces this data, and further processing steps (e.g., clustering, pyramid generation) consume it.

Although a `PointSetList` provides a natural partitioning of the data points into

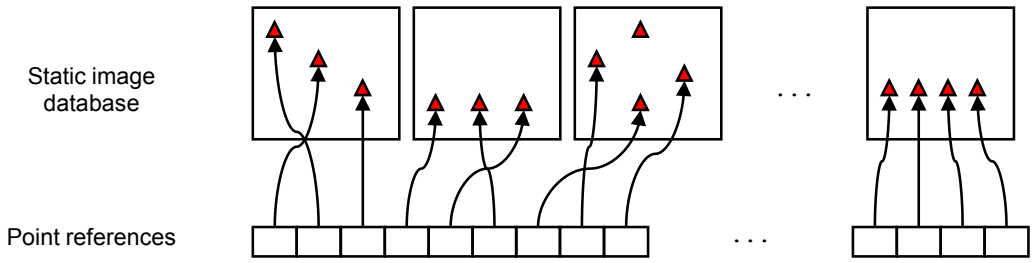


Figure 5-1: Two representations of a LIBPMK image set. The top representation divides the feature points into groups based on which image they are from; the bottom allows for arbitrary groupings while avoiding the creation of duplicate copies of the database.

logical groups (one group per image), certain operations, like clustering, ignore these boundaries. Such operations may want to regroup or otherwise manipulate the ordering of the points. Figure 5-1 shows an alternate representation of the image database that may be used for this purpose. Rather than duplicating feature points in memory (which may be high-dimensional), the data is given as a mutable array of references to the `PointSetList` representation.

Because large image sets can pose problems if they contain too much data to fit in memory at once, LIBPMK provides a transparent interface to `PointSetList` structures that reside on disk. The on-disk interface tries to minimize disk seeks by maintaining two caches of `PointSet` structures: a lookahead array and a least-recently-used (LRU) cache. If there is ever a query that misses both caches, the lookahead array is populated with the next few `PointSet` objects, and the LRU cache is updated appropriately.

The optimal size of each of these caches is determined by the application: for applications that read the images sequentially, a larger lookahead array is suitable and the LRU cache is insignificant; for applications with random-access behavior, the best option is to keep the lookahead array small and the LRU cache large. Figure 5-2 illustrates the layout in memory of the on-disk `PointSetList` interface. Because this scheme is transparent and interchangeable with the scheme that stores the entire image set in memory, developers can switch between them so that small experiments

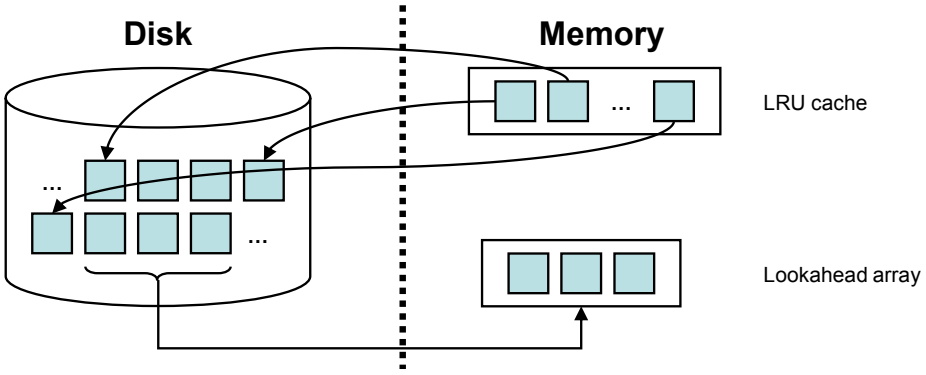


Figure 5-2: Although the interface remains the same as one kept entirely in memory, a `PointSetList` kept on disk maintains two caches to minimize the number of disk seeks required. Each shaded box represents one image.

can easily be scaled up to large applications effortlessly.

Pyramids

Hierarchical structures are commonly used in many object recognition and image retrieval algorithms. Of these, LIBPMK makes the most use of pyramids, which are varying-resolution histograms made either of grids imposed on the feature space or a partition defined by hierarchical clustering (such as a vocabulary tree).

Pyramids are represented in LIBPMK as sparse trees. A node in a tree is uniquely identified by a set of small integers describing a path through the tree, and has pointers to its parent node as well as the head and tail of a linked list representing the children. The child nodes are represented in this manner so that insertions and deletions are handled quickly (the branch factor of the tree is not fixed, not only because the tree is sparse, but because the maximum number of children of any node does not have to be specified). Because the points in a single image only occupy small parts of the feature space, most of the tree is empty, and thus a sparse representation is beneficial here. With this representation, we can perform pyramid match computations by traversing the tree exactly once; because it is sparse, the running time is therefore bounded by the number of feature points in the image.

LIBPMK does not distinguish between vocabulary-based partitions of the feature space and uniform-grid partitions. The uniform grid forms a special case, in which

it is unnecessary to record the parent-child relationships between nodes. As a result, it is possible to form pyramids just by quantizing each feature point [10]. However, LIBPMK instead chooses the slightly slower approach of explicitly creating the tree structure and traversing it in order to maintain flexibility: whether the pyramid structure is defined by uniform grids, by clustering, or by any other method, they are interchangeable with one another, and thus the pyramid matches for all of these cases can be computed with the same processing pipeline. Section 5.2.2 describes the general framework of the pyramid match pipeline.

5.2.2 System architecture

As outlined in Section 4.1.1, vision systems have similar processing pipelines. LIBPMK implements each step of the pipeline as an independent module, allowing developers to insert modifications at any step with ease. The following is a description of the LIBPMK processing pipeline for a typical vocabulary-guided pyramid match:

1. Extract features (e.g., SIFT [17], geometric blur [2], shape context [1]) from the training images, resulting in a `PointSetList`.

2. Perform hierarchical clustering over a sample of the points to produce a hierarchical vocabulary. This vocabulary tree may be used directly for indexing tasks such as in [21].

3. For each training image, generate its sparse pyramid representation.

4. Perform the pyramid match on every pair of training images to form a kernel, which will then be used to train an SVM.

Figure 5-3 summarizes the processing pipeline described above. This architecture assumes that all of the training images are presented before any processing is done. In order for the system to update itself with new training data, the entire computation must be performed again from the start, which is extremely costly.

Figure 5-4 shows a modified version of the system, which uses the adaptive vocabulary forest method from Chapter 3 and the VT-SVM from Chapter 4. Here, the

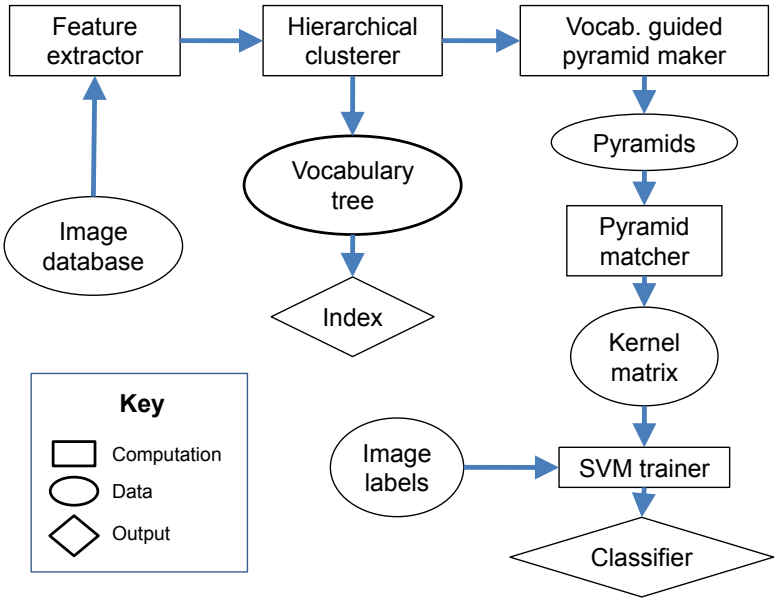


Figure 5-3: A typical processing pipeline for a vocabulary-guided pyramid match. The output of the clustering module can be used as a vocabulary tree for indexing tasks, or the data can continue through the pyramid/SVM pipeline to build an SVM classifier. The entire computation is performed offline.

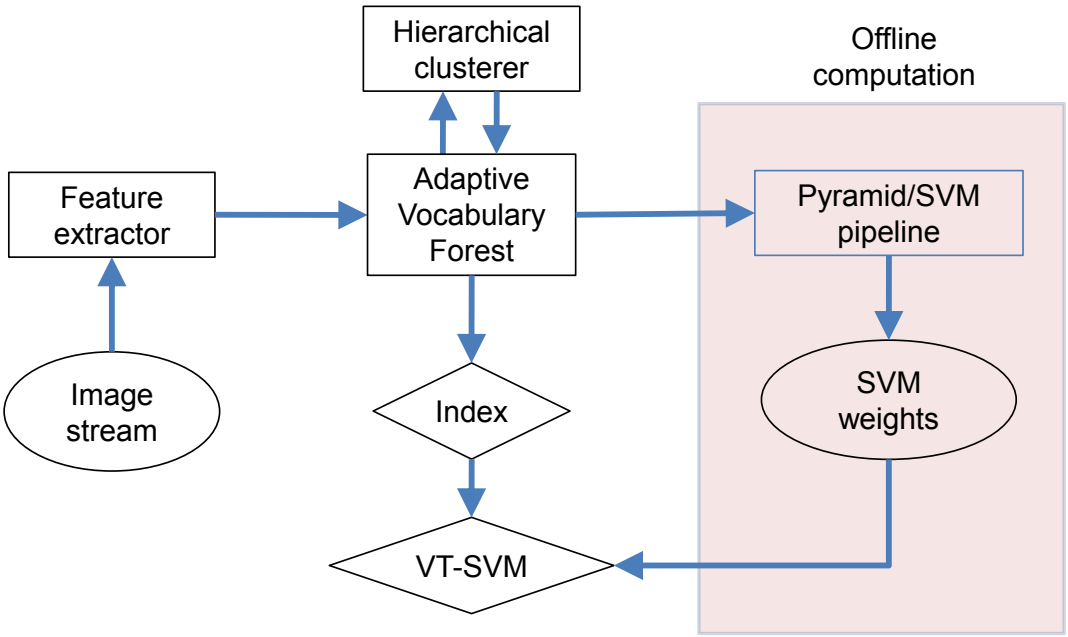


Figure 5-4: The processing pipeline for an online system using both the adaptive vocabulary forest method (Chapter 3) and VT-SVM (Chapter 4). The offline computations can adjust the VT-SVM, but do not otherwise affect its normal operation. The pyramid/SVM pipeline here is the same as the right-most column of Figure 5-3.

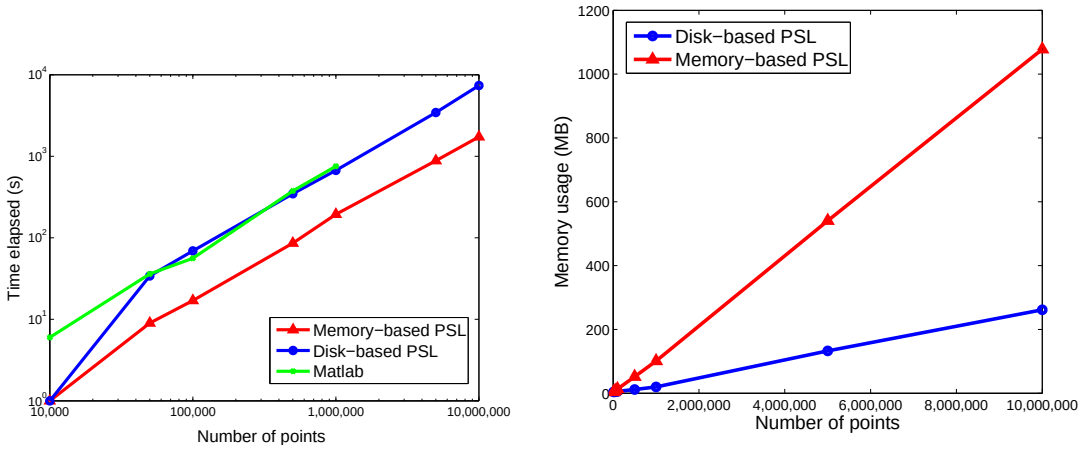
images are shown to the system as a *stream*, revealing the properties that (1) queries and updates are performed online and (2) the order of images is significant, since the adaptive method can capture trends. Furthermore, the VT-SVM component can be tweaked over time by an offline process that runs on the side, which is useful if object categories change dramatically over time. Figure 5-4 also shows the versatility of the adaptive vocabulary forest, as it is able to produce an index for content-based retrieval tasks in addition to classifiers for category-learning tasks.

The modularity of the system is of great importance. Observe that between Figures 5-3 and 5-4, many of the component boxes transfer from one system to the other, despite the fact that one system is static while the other operates adaptively. With minimal code, developers can make fast prototypes of radically different systems by simply reusing modules wherever possible. LIBPMK offers not only this flexibility, but also efficiency, as described in the next section.

5.3 Evaluation

I evaluated the performance of LIBPMK by analyzing the processor and memory usage of one of the most computationally extensive tasks in the package, k-means clustering. The clusterer is used to quantize features into vocabularies, and is also used as a submodule in the hierarchical clustering methods that build vocabulary trees. This experiment used seven synthetically-generated datasets of varying size. Each dataset contained 12-dimensional features, with 100 features per image – typical settings for most algorithms. The data sets varied in size from 100 images to 100,000 images; thus, the experiments shown in this section operate on datasets consisting of up to 10 million points. All experiments were run on a 3.2 GHz desktop with 2 GB of RAM.

Each dataset was clustered using $k = 10$, using both the memory-based `PointSetList` (PSL) and the disk-based PSL; the same data was also loaded into MATLAB, a standard tool used by many vision researchers, and clustered using the built-in k-means functionality. The speed here is reported as the time taken to complete 100



(a) Speed

(b) Space usage

Figure 5-5: K-means clustering performance with LIBPMK. The memory-based database is faster than the disk-based representation, but consumes much more memory. The MATLAB implementation is as slow as the disk-based LIBPMK database and consumes at least as much space as the memory-based database; databases of size greater than 1 million could not be used with MATLAB due to hardware constraints.

iterations of k-means. Because of differences in platform, the space usage between MATLAB and LIBPMK cannot be fairly compared, as MATLAB loads other, possibly unrelated data into memory before even starting the operation. In practice, the MATLAB implementation consumes as much memory as the memory-based PSL, plus a fixed cost of nearly 400MB. This gives severe limitations on the kinds of data sets that can be processed.

Figure 5-5 shows both the speed and memory usage for this task. Because the MATLAB implementation was unable to handle more than 1 million points, the curve beyond that point is not shown. The figures also show that implementers can choose to make trade-offs depending on application: the memory-based PSL clearly consumes more memory (at least as much as the data set itself consumes on disk), but runs an order of magnitude faster. On the other hand, the disk-based PSL uses significantly less memory and is able to work with extremely large data sets.

Although, for the specific task of clustering, the problem of space limitations for MATLAB is generally mitigated by sampling the data points, it shows that such a setup can not be effectively used seamlessly in an end-to-end system. As flexibility

is one of the most important goals of LIBPMK, its ability to operate should be independent of the size or organization of the input data, so that no development time is consumed tediously watching over the execution of each step.

5.4 Applications built with LIBPMK

This section describes three applications that were built using LIBPMK: mobile content-based image retrieval, photo-oriented question answering, and an experimental framework. The diversity of these applications show the many uses of LIBPMK for computer vision tasks on various platforms. A manual for LIBPMK is provided in [15].

5.4.1 Mobile content-based image retrieval

This application was built to demonstrate the properties of the adaptive vocabulary forest described in Chapter 3. In this demo, a user takes a picture with a camera phone. The photo is then uploaded to a nearby computer via Bluetooth, where it enters the vocabulary forest pipeline (as in Figure 5-4) to retrieve the top n matching images. The matching images are then displayed on the screen; the query image is also added to the database immediately, so that future queries will be able to retrieve that image.

This system was built to be run from a laptop computer; on a 1 GHz processor and 512 MB of RAM, the system runs through the entire pipeline in only a few seconds (with the main bottleneck being the I/O involved in converting between image formats) with hundreds of images in the database, allowing for responsive user interaction. Although designed to operate as a toy example, this application highlights the adaptability and incremental nature of the adaptive vocabulary forest method.

Figure 5-6: A photo-oriented question answering system built with LIBPMK as the image matching engine. In steps 1 and 2, a user takes a photo and asks a question about it. In steps 3, 4, and 5, the user browses the matching images, other questions about them, and their answers, respectively. Steps 6 and 7 show a user asking a question about an existing photo and receiving a direct response.

5.4.2 Photo-oriented question answering

A photo-oriented question (POQ) is one in which a text question is accompanied with an image. This scheme allows for the question to refer to something more descriptive than what could be written as text; this is generally most useful for queries about visually distinctive or oddly-shaped objects or about specific instances. Additionally, users can give more succinct questions if there is visual context, making this application even more practical on mobile devices.

A POQ system was implemented using LIBPMK in [29], where the image matching engine of LIBPMK was combined with a question matching engine to populate entries in a multimodal query database. On one end, a user would submit a POQ to search the database, and an expert community on the other end populated the database (in a manner similar to current Web-based question-answering systems such as Yahoo! Answers). The adaptive vocabulary forest of Chapter 3 was used as the image matching engine, while a simple TF-IDF metric was used for matching the questions. This particular system contained images of buildings at MIT, and was aimed at tourists who were curious about MIT. Test subjects were found to ask questions such as "who is the architect of this building?" and "when was this building built?" using the photo in place of a textual description.

The end-to-end query time for this system – from the user taking the photo on a phone to getting a response via SMS – was about 30 seconds for a system supporting

15 simultaneous users. As expected, the largest bottleneck is the time involved with sending the SMS message and waiting to receive it (operator-dependent) and the I/O involved with decoding and encoding the image files.

5.4.3 Automated testing

LIBPMK is used not only to build interactive applications but also for easy automated experimentation on data sets commonly used for research purposes, such as Caltech-101 [9] Caltech-256 [12]. Unlike the two applications described above, the procedures in this setting work in batch rather than with one image at a time. Although the processing pipelines are the same as shown in Figures 5-3, the entire data set is pushed through each node rather than sending them one at a time, reducing the overhead associated with a large number of small I/O operations.

The flexibility of LIBPMK allows for easy extension to other experimental setups. Generally, the desired output is a kernel matrix comparing all training and test examples with one another, which is then used as input to train and test an SVM; LIBPMK integrates with LIBSVM [5], a well-known SVM library, to perform this task. The spatial pyramid matching of [14] is implemented easily within this framework by simply using spatial features with quantized appearance into a set of pyramid-matchers; a kernel is generated with no other changes to the processing pipeline. Similarly, the "extremely randomized clustering forest" approach of [20] was implemented using LIBPMK simply by replacing the clustering module with a random partitioner.

LIBPMK was used as the visual recognition component of [23], which combined speech classifiers with image classifiers to obtain improved results on the Caltech-101 database. LIBPMK was also used in [8] to build sets of feature histograms for multi-view images, which were successfully used to perform distributed feature selection to encode data from several cameras. The wide variety of LIBPMK's experimental applications highlights its flexibility and extensibility.

Chapter 6

Conclusion

In this thesis I have described some of the major requirements of a user-oriented dynamic object recognition system that supports many images and many users: speed, adaptability, and scalability. I have shown that although many existing algorithms achieve good speed and accuracy, they may encounter problems when scaling up to larger applications. The main contributions of this thesis are new algorithms designed specifically for such environments.

I have introduced efficient techniques for object recognition and content-based image retrieval tasks in interactive dynamic environments. These algorithms take advantage of the speed and accuracy of vocabulary trees while also adapting to trends in images and scaling up robustly. I evaluated these techniques on several data sets and demonstrated that they can be successfully applied to a variety of computer vision tasks.

The adaptive vocabulary forests introduced here are not only faster, but achieve the same accuracy as the slower batch methods do. Because the trees are grown incrementally, they can quickly alter their structure to perform better on recent classes of images. I have shown that pruning the trees also leads to an extremely space-efficient representation that outperforms larger, static trees. Extensive experimentation with the parameters of the adaptive vocabulary forests shows that the recognition accuracy is fairly insensitive to them, allowing for an extremely versatile time-space tradeoff that makes it suitable for many applications and architectures.

I have also introduced the VT-SVM, which combines the efficiency of vocabulary trees with the recognition ability of an SVM. I showed that by using only 10% of the training examples, the VT-SVM is able to match the results of a standard SVM, providing nearly a ten-fold speed increase in query time. This method allows users to choose a fixed number of image comparisons to make per query. Because this is independent of the size of the training set, this method scales easily to larger databases.

I have developed LIBPMK for the purpose of enabling developers to quickly create new applications easily, as well as larger-scale experiments for research settings. The techniques and the implementations I have developed can be used for a variety of applications, such as example-based photo grouping, content-based tag suggestion, and category-based image searching. They are well-suited to such applications because of their efficiency in large-scale dynamic environments, in which algorithms need to not only be fast, but also adaptable and scalable.

Recommendations for Future Work

While this thesis has laid some groundwork for practical dynamic vision systems, much work remains to be done. While the current systems can easily handle hundreds of thousands of images with reasonable speed, the accuracy of any of these algorithms declines as the database size grows. As a result, these algorithms become less viable for practical use when operating on the scale of millions or billions of images. Additionally, because current experiments generally use data sets small enough to fit on a single machine, many algorithms are evaluated on images that are not necessarily a representative set of all images accessible to us.

Much recent work has been focused in combining many different recognition algorithms (weighted appropriately) to improve the accuracy. Although each of these methods individually may have its share of problems when applied to the dynamic environments described in this thesis, the performance of the combination of these methods on static data sets is very promising. To successfully transfer these benefits

to user-oriented applications, further work would need to be done in optimizing their speed and scalability. Speed, in particular, is an important issue, because algorithms based on a weighted combination of other methods are, by definition, at least as slow as all of its constituent algorithms combined.

In addition to algorithmic challenges involved with Internet-scale applications, we face many problems with system design. Servers must be able to handle the enormous load of many simultaneous queries, each of which may need to adapt the system. To be tolerant to failures, the system needs to be replicated, and we need to design methods for keeping all replicas up-to-date.

Bibliography

[1] Serge Belongie, Jitendra Malik, and Jan Puzicha. Shape context: A new descriptor for shape matching and object recognition. In *Advances in Neural Information Processing Systems (NIPS)*, pages 831–837, 2000.

[2] Alexander C. Berg and Jitendra Malik. Geometric blur for template matching. In *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 607–614, 2001.

[3] Anna Bosch, Andrew Zisserman, and Xavier Munoz. Representing shape with a spatial pyramid kernel. In *Proceedings of the 2007 ACM International Conference on Image and Video Retrieval (CIVR)*, 2007.

[4] Gert Cauwenberghs and Tomas Poggio. Incremental and decremental support vector machine learning. In *Advances in Neural Information Processing Systems (NIPS)*, 2000.

[5] Chih-Chung Chang and Chih-Jen Lin. *LIBSVM: a library for support vector machines*, 2001. Software available at http://www.csie.ntu.edu.tw/ cjlin/libsvm.

[6] Moses Charikar, Chandra Chekuri, Tomas Feder, and Rajeev Motwani. Incremental clustering and dynamic information retrieval. In *Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of Computing (STOC)*, pages 626–635. ACM Press, 1997.

[7] Wai chiu Wong and Ada Wai chee Fu. Incremental document clustering for web page classification. In *International Conference on Information Society*, Japan, 2000.

[8] C. Mario Christoudias, Raquel Urtasun, and Trevor Darrell. Unsupervised distributed feature selection for multi-view object recognition. Technical Report MIT-CSAIL-TR-2008-009, MIT Computer Science and Artificial Intelligence Laboratory, April 2008.

[9] Li Fei-Fei, Rob Fergus, and Pietro Perona. Learning generative visual models from few training examples: An incremental bayesian approach tested on 101 object categories. *IEEE Computer Vision and Pattern Recognition (CVPR) Workshop of Generative Model Based Vision*, 12:178, 2004.

[10] Kristen Grauman and Trevor Darrell. The pyramid match kernel: Discriminative classification with sets of image features. In *Proceedings of the Tenth IEEE International Conference on Computer Vision (ICCV)*, pages 1458–1465, Washington, DC, USA, 2005. IEEE Computer Society.

[11] Kristen Grauman and Trevor Darrell. Approximate correspondences in high dimensions. In *Advances in Neural Information Processing Systems (NIPS)*, 2006.

[12] Greg Griffin, Alex Holub, and Pietro Perona. Caltech-256 object category dataset. Technical Report 7694, California Institute of Technology, 2007.

[13] Yan Ke and Rahul Sukthankar. PCA-SIFT: a more distinctive representation for local image descriptors. In *Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, volume 2, pages 506–513, 2004.

[14] Svetlana Lazebnik, Cordelia Schmid, and Jean Ponce. Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories. In *Proceedings of the 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, 2006.

[15] John J. Lee. Libpmk: A pyramid match toolkit. Technical Report MIT-CSAIL-TR-2008-017, MIT Computer Science and Artificial Intelligence Laboratory, April 2008.

[16] Vincent Lepetit. Keypoint recognition using randomized trees. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28(9):1465–1479, 2006. Member-Pascal Fua.

[17] David G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision (IJCV)*, 60(2):91–110, 2004.

[18] Jiri Matas, Ondrej Chum, Martin Urban, and Tomas Pajdla. Robust wide baseline stereo from maximally stable extremal regions. In *Proceedings of the British Machine Vision Conference (BMVC)*, volume 1, pages 384–393, London, 2002.

[19] Krystian Mikolajczyk and Cordelia Schmid. Scale & affine invariant interest point detectors. *International Journal of Computer Vision (IJCV)*, 60(1):63–86, 2004.

[20] Frank Moosmann, Bill Triggs, and Frederic Jurie. Fast discriminative visual codebooks using randomized clustering forests. In *Advances in Neural Information Processing Systems (NIPS)*, 2006.

[21] David Nister and Henrik Stewenius. Scalable recognition with a vocabulary tree. In *Proceedings of the 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2161–2168, Washington, DC, USA, 2006. IEEE Computer Society.

[22] Arnaud Ribert, Abdel Ennaji, and Yves Lecourtier. An incremental hierarchical clustering. In *Proceedings of the IAPR 15th International Conference on Vision Interface (VI)*, pages 586–591, 1999.

[23] Kate Saenko and Trevor Darrell. Towards adaptive object recognition for situated human-computer interaction. In *WMISI '07: Proceedings of the 2007 workshop on Multimodal interfaces in semantic interaction*, pages 43–46, New York, NY, USA, 2007. ACM.

[24] Hao Shao, Tomas Svoboda, and Luc Van Gool. Zubud – zurich buildings database for image based recognition. Technical Report 260, Swiss Federal Institute of Technology, 2004.

[25] Josef Sivic and Andrew Zisserman. Video google: A text retrieval approach to object matching in videos. In *Proceedings of the Ninth IEEE International Conference on Computer Vision (ICCV)*, page 1470, Washington, DC, USA, 2003. IEEE Computer Society.

[26] Manik Varma and Debajyoti Ray. Learning the discriminative power-invariance trade-off. In *Proceedings of the Eleventh IEEE International Conference on Computer Vision (ICCV)*, 2007.

[27] Christian Wallraven, Barbara Caputo, and Arnulf Graf. Recognition with local features: the kernel recipe. In *Proceedings of the Ninth IEEE International Conference on Computer Vision (ICCV)*, page 1470, Washington, DC, USA, 2003. IEEE Computer Society.

[28] Tom Yeh and Trevor Darrell. Dynamic visual category learning. In *Proceedings of the 2008 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, 2008.

[29] Tom Yeh and Trevor Darrell. Multimodal question answering for mobile devices. In *Proceedings of the 2008 ACM International Conference on Intelligent User Interfaces (IUI)*, pages 405–408, 2008.

[30] Hao Zhang, Alexander C. Berg, Michael Maire, and Jitendra Malik. Svm-knn: Discriminative nearest neighbor classification for visual category recognition. In *Proceedings of the 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2126–2136, 2006.

[31] Tian Zhang, Raghu Ramakrishnan, and Miron Livny. Birch: an efficient data clustering method for very large databases. *ACM SIGMOD Record*, 25(2):103–114, 1996.